공학박사학위논문

Computational Approaches on Choreographing Multiple Actor Motion

컴퓨터를 활용한 여러 사람의 동작 연출

2017 년 8 월

서울대학교 대학원

전기.컴퓨터 공학부

원 정 담

Abstract

Choreographing motion is the process of converting written stories or messages into the real movement of actors. In performances or movie, directors spend a considerable time and effort because it is the primary factor that audiences concentrate. If multiple actors exist in the scene, choreography becomes more challenging. The fundamental difficulty is that the coordination between actors should precisely be adjusted. Spatio-temporal coordination is the first requirement that must be satisfied, and causality/mood are also another important coordinations. Directors use several assistant tools such as storyboards or roughly crafted 3D animations, which can visualize the flow of movements, to organize ideas or to explain them to actors. However, it is difficult to use the tools because artistry and considerable training effort are required. It also doesn't have ability to give any suggestions or feedbacks. Finally, the amount of manual labor increases exponentially as the number of actor increases.

In this thesis, we propose computational approaches on choreographing multiple actor motion. The ultimate goal is to enable novice users easily to generate motions of multiple actors without substantial effort. We first show an approach to generate motions for shadow theatre, where actors should carefully collaborate to achieve the same goal. The results are comparable to ones that are made by professional actors. In the next, we present an interactive animation system for pre-visualization, where users exploits an intuitive graphical interface for scene description. Given a description, the system can generate motions for the characters in the scene that match the description. Finally, we propose two controller designs (combining regression with trajectory optimization, evolutionary deep reinforcement learning) for physically simulated actors, which guarantee physical validity of the resultant motions.

Keywords: Graphics, Character Animation, Multiple Actor, Choreography, Authoring, Physics-based Control, Deep Learning, Reinforcement Learning

Student Number: 2011-20881

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Choreographing motion is the process of converting written stories or messages into the real movement of actors. In performances or movie productions, directors spend a considerable time and effort to design motions of actors because it is the primary factor on which audiences focus a lot. The director usually has several concepts/ideas on the scene, he first explains them to actors. The actors then perform in their own ways according to the requirements, and the director again talks about wrong or unsatisfying parts (see Figure 1.1). The process is repeated until resultant motions are satisfactory. The process, as a result, requires a lot of prior knowledge, abundant experiences, high creativity, and good communication skills.

If multiple actors exist in the scene, choreographing their motions becomes more complex and time-consuming. The fundamental challenge is that the coordination between actors should precisely be adjusted. The coordination can be explained in several point of views. Spatio-temporal coordination is the minimal requirement that must be satisfied when creating multiple actor motion. For example, If two actors are fighting, it is crucial to precisely adjust their relative positions and timing of the hitting moment. If the coordinations are not satisfied, then we could have unrealistic motions. One punches in the air and the other falls alone, the timing furthermore


Figure 1.1 Still pictures of choreographing motions in a real performance/movie. (Left) A performance *Poor People's TV Room*: ©The Andrew W. Mellon Foundation . (Right) A movie *The Monuments Men*: ©Columbia Pictures.

may not be matched. Causality and mood are also another important coordinations. Causality between motions gives the logical plausibility, whereas fitting mood of motions gives the emotional plausibility. Because these are subtle and subjective factors, only a small subset of motions looks plausible among all possible combinations. The complexity increases exponentially whenever a new actor participates in the scene.

Directors use several assistant tools, which can visualize the flow of movements, to organize ideas or to explain them to actors/investors. The representative one is a storyboard (see Figure 1.2). A storyboard is a sequence of illustrations or images of several representative postures in the scene, where short written explanations could be added if necessary. 3D animated pre-visualization have also been used for performances or movies produced by huge production companies (see Figure 1.3). This is for reproducing the final result at a rough level in advance. The two animation-based visualizations enable us to watch all actors' motions simultaneously, which makes it easier to choreograph multiple actor motion. However, it is difficult to use the tools freely due to several reasons. First, it requires artistry and considerable training effort, only professionals can use the tools as a result. Second, they are inherently passive tools, which means that it can not give any suggestions or feedbacks when users do something wrong or need assists when choreographing. Finally, it is not scalable. The amount of manual labor increases exponentially as the number of actor appearing in

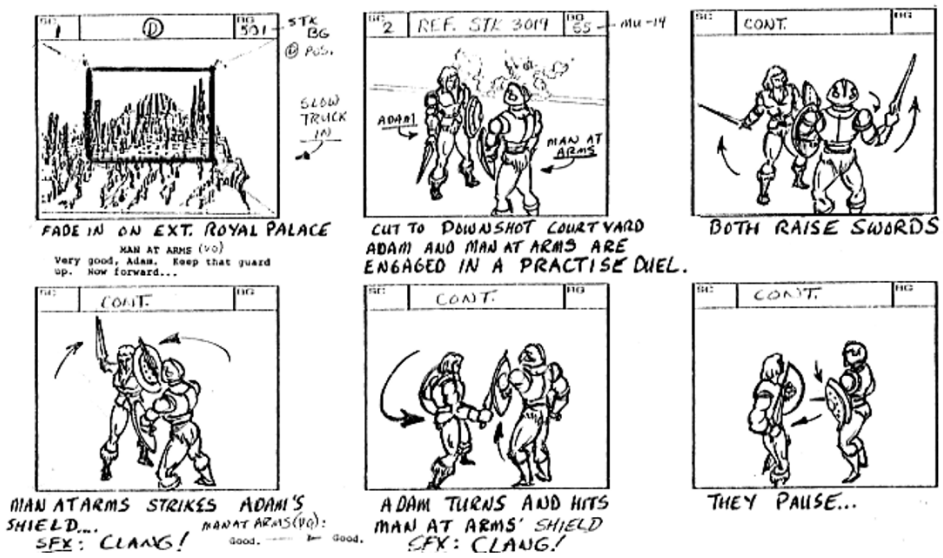


Figure 1.2 A storyboard of *The Return of Granamyr*: ©Filmation Associates. Two gladiators motions are illustrated by their representative postures, arrows around, and a few sentences.

the scene increases.

In this thesis, we propose computational approaches on choreographing multiple actor motion. The ultimate goal is to enable novice users easily to generate motions of multiple actors which satisfy key coordinations (space, time, causality, and mood) without substantial effort. The first part of the thesis introduces an approach to generate motions for shadow theatre, where actors should carefully collaborate so that a whole shadow of them matches to the target shadow image. This requires a lot of creativity and special stage settings. In the second part, we present an interactive animation system for pre-visualization. By exploiting an intuitive graphical interface for scene descriptions, users can easily generate motions for the scene in which complex interactions exist. Although the two parts tackled several challenges in choreographing multiple actor motion, they are inherently kinematic approaches. This means that substantial computation is required to ensure physical plausibility of the resultant motions entirely. Modeling actors in physically simulated environments and designing

Figure 1.3 A pre-visualized animation of *Pirates of The Caribbean*: ©LFA Previsualization.

controllers for them could be one promising approach because physics simulation guarantees physical validity of generated motions. In the final part of the thesis, we present an approach that adopts this framework to generate motions. We especially choose bird-like non-human actors as an example, who use flapping flight as a primary motor skill.

Motion Generation for Shadow Theatre: Shadow theatre is a genre of performance art in which the actors are only visible as shadows projected on the screen. The goal is to generate animated characters, the shadows of which match a sequence of target silhouettes. This poses several challenges. The motion of multiple characters are carefully coordinated to form a target silhouette on the screen, and each character's pose should be stable, balanced, and plausible. The resulting character animation should be smooth and coherent spatially and temporally. We formulate the problem as nonlinear constrained optimization with objectives, which were designed to generate plausible human motions. The optimization algorithm was primarily inspired by the heuristic strategies of professional shadow theatre actors. Their know-how was studied and then incorporated into the optimization formulation. We demonstrate the effectiveness of our approach with a variety of target silhouettes and 3D fabrication of the results (see Figure 1.4).

Figure 1.4 From a sequence of 2D silhouette images, our shadow theatre system generates animated characters, of which projection on the screen matches well with the target shapes. Our approach is applicable to various shapes such as an elephant (yellow), a rabbit (green), a car (red), and a violin (blue).

Interactive Animation System for Pre-visualization: In many application areas, such as animation for pre-visualizing movie sequences and choreography for dance or other types of performance, only a high-level description of the desired scene is provided as input, either written or verbal. Such sparsity, however, lends itself well to the creative process, as the choreographer, animator or director can be given more choice and control of the final scene. Animating scenes with multi-character interactions can be a particularly complex process, as there are many different constraints to enforce and actions to synchronize. Our novel 'generate-and-rank' approach rapidly and semi-automatically generates data-driven multi-character interaction scenes from high-level graphical descriptions composed of simple clauses and phrases. From a database of captured motions, we generate a multitude of plausible candidate scenes. We then efficiently and intelligently rank these scenes in order to recommend a small but high-quality and diverse selection to the user. This set can then be refined by re-ranking or by generating alternatives to specific interactions. While the approach is applicable to any scenes that depict multi-character interactions, we demonstrate its efficacy for choreographing fighting scenes and evaluate it in terms of performance and the diversity and coverage of the results (see Figure 1.5).

Figure 1.5 We animate data-driven scenes with multi-character interactions from high-level descriptions. Many plausible scenes are generated and efficiently ranked, so that a small, diverse and high-quality selection can be presented to the user and iteratively refined.

Controller Design for Flapping Flight: Designing physics-based controllers for a flying creature is very challenging particularly when the dynamic model of the creatures is high-dimensional, having many degrees of freedom. We present a control method for flying creatures, which are aerodynamically simulated, interactively controllable. Two methods that construct state-action mapping are introduced in particular: The first method generates state-action pair data by running a collection of trajectory optimizations for a cycle of wingbeat, then builds KNN (K Nearest Neighbor) regression model from the data. Grid models and a speedup algorithm for running a collection of similar trajectory optimization problems are introduced. The second method represents the mapping as Deep Neural Networks (DNN) and learns it using Deep Q-Learning (DQL). The control method is example-guided in the sense that it provides the user with direct control over the learning process by allowing the user to specify keyframes of motor skills. We suggests novel learning algorithm, which was inspired by evolutionary strategies of Covariance Matrix Adaptation Evolution Strategy (CMA-ES), to improve the convergence rate and the final quality of the control policy. The effectiveness of the two method is demonstrated with imaginary winged creatures flying in a physically simulated environment and their motor skills that achieves goals such as reaching a target location or following user-provided paths without losing a balance.

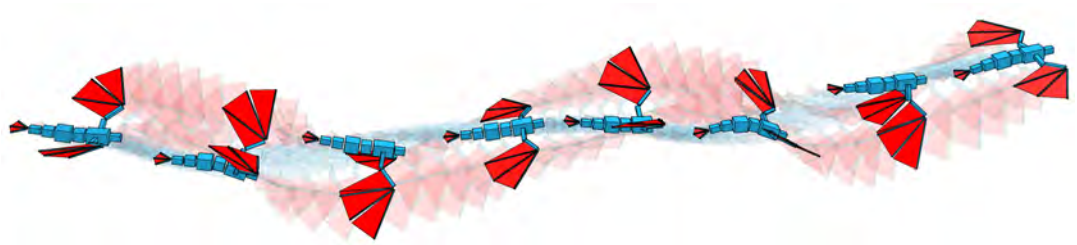


Figure 1.6 The imaginary dragon learned to fly through repeated experiences. The dragon is physically simulated in realtime and interactively controllable.

The remainder of the thesis is organized as follows. Chapter 2 introduces previous studies that are related to the thesis, Chapter 3 describes motion generation approach for shadow theatre performance, Chapter 4 demonstrates an interactive animation system for pre-visualization by showing several scenarios and their synthesized result scenes. Chapter 5 presents two controller designs for physically simulated actors. Finally, Chapter 6 concludes the thesis and describes the future direction of computational approaches for choreographing multiple actor motion.

Chapter 2

Background

This chapter gives you a brief explanation of previous studies to understand the thesis. In the first section, several fundamental techniques to generate character motions including editing and synthesis for single/multi character animation, motion planning for satisfying spatio-temporal constraints, motion control by reinforcement learning, pose/motion estimation from incomplete information, and diversity on motion generation. In the second section, several authoring systems for animation purpose will be presented. There are two categories: One generates motions from high-level (abstract) inputs, the other focuses tunable output to meet the preference of users. In the third section, researches on shadows generation/exploitation will be introduced to understand the shadow theatre performance. In the final section, several controller design methodologies for physically simulated characters including trajectory optimization, model-based control, direct policy learning, and deep reinforcement learning.

2.1 Motion Generation Technique

2.1.1 Motion Editing and Synthesis for Single-Character

Generating realistic movements for human-like characters have been extensively studied in computer graphics community. Because it is hard to generate them from scratch, many researchers have exploited motion capture data as a basis then edit or synthesize new motions that match to desired situations. The main goal is to preserve naturalness in the original data while manipulating them as freely as possible. Lee et al. [53] analyzed the frequency of existing motions data and editing them from low-frequency domain to high-frequency domain, which accelerated computational speed a lot. Blending-based researches have also been studied. Kovar et al. [46] proposed a distance metric for logically similar but numerically different cases, and parameterized similar data in an automatic manner. Mukai et al. [80] viewed motion blending problem as statistical predictions of missing data, then they optimize interpolation kernels by analyzing the correlation of the data. There have been studies that the motion data first are segmented into short-term clips, then the clips are rearranged in an order of satisfying constraints [54, 47, 6, 92].

2.1.2 Motion Editing and Synthesis for Multi-Character

It is difficult to handle motions when multiple characters exist due to not only the number of character but also the interactions between them. There have been researches to solve the issues. Kim et al. [44] uses Laplacian motion editing to allow the user to interactively manipulate synchronized multi-actor motions. Ho et al. [35] use an *Interaction Mesh* to edit and retarget close interactions between body parts of single or multiple characters, while Shum et al. [97] apply min-max searching to produce scenes of two character interactions to follow multiple story-lines. Game theory and motion graphs have also been used to control the motion of two characters in an adversarial game [110]. The scope of motion synthesis algorithms based on min-max searching and adversarial games are often limited to interactions between two

characters.

Many scalable multi-character motion synthesis algorithms make use of *Motion Patches*. The original work by Lee et al. [55] tackles the problem of capturing motion data for a large and complex virtual environment by constructing a set of building blocks that can be arbitrarily assembled to create novel environments. Each block is annotated with a *Motion Patch*, which describes the animations available for characters within that block. Even though the original work explores character animation in complex environments, its followup studies focus on creating a dense crowd of interacting characters. Crowd Patches involves a similar approach [119] for large-scale urban scenes. Support has also been provided to simulate close interactions between characters, using tileable *Interaction Patches* [98]. However, fully connecting scattered interaction patches when the connectivity is not simple (i.e., with cycles) is difficult. Patch tiling by Kim et al [45] and an MCMC method by Hyun et al [37] generate fully-connected (spatially coordinated and temporally synchronized) interaction scenes, with the focus on removing all dead-end connections. Resulting scenes tended to be random rather than controlled and detailed scene descriptions could not be satisfied.

2.1.3 Motion Planning

Satisfying spatio-temporal constraints is a crucial requirement that gives a lot of freedom to users in motion generation. There have been studies that approach the problem in a path planing point of view. With path planning methods, only a start and goal are provided for the character, and the system must automatically move them along the generated trajectory. Such motion controllers can be learned based on motion capture data [69], and parameterized to correspond to specific motion skills using reinforcement learning [63]. Choi et al. [13] generate and traverse a graph of motion capture fragments, which may be deformable in order to situate them into highly constrained virtual environments. Other sparse descriptions include simple motion sketches that are satisfied by efficiently searching through a motion graph and

generating the interpolated motion trajectory [92], or a timeline painted with simple annotations selected from a domain-tailored vocabulary, again using optimized searching to deliver interactive authoring [6]. These approaches do not however efficiently provide a high level of scene variations while satisfying the constraints imposed by multi-character interactions.

2.1.4 Motion Control by Reinforcement Learning

Recently, reinforcement learning have gained a lot of attention due to its generalization capability. Reinforcement learning assumes a Markovian decision process. Control policy π decides which action to take at any state s, which then result in transitioning to a subsequent state s' and a reward r. Reinforcement learning generates an optimal control policy that maximizes the sum of expected future rewards, meaning that future plans are taken into account for the control policy. The expected future rewards and the control policy are often represented as function approximators, which are called value and policy functions, respectively. Reinforcement learning has been used successfully for designing character controllers with motion capture data [52, 107, 57] and for planning the motion of physically simulated bipeds [14].

2.1.5 Pose/Motion Estimation from Incomplete Information

Because it is challenging to generate pose/motion from scratch, there have been extensive approaches utilizing additional information which is usually 2D and incomplete (e.g., sketches, silhouettes). One common approach is to construct a mapping function from 2D information into original 3D poses using human motion databases [54, 1, 90, 89, 21, 96]. Another class of approaches is directly extracting poses by minimizing the visual differences between given 2D inputs and the projection of estimated 3D poses [100, 19, 39, 27, 113, 65, 91, 28, 12].

2.1.6 Diversity on Resultant Motions

Several studies have considered the diversity of the results when generating motions. The diversity could increase the probability satisfying users with the results. Agrawal et al. [2] present an optimization framework for generating diverse variations of physics-based character motions, while Liu et al. [67] use randomized sampling and forward dynamics simulation to reconstruct many variations on a given motion capture trajectory. Ye et al. [118] also use randomized sampling for synthesizing a variety of physically simulated hand motions. Jain et al. [39] present an interactive motion editing tool for creating dynamic scenes with human and object interaction, which allows the animator to directly change the trajectories of the human or objects and simultaneously render the effect of the edits on the entire scene. Optimization has also been used to synthesize more complex human behaviors, including cooperative actions involving multiple characters and their interactions with their environment and each other [3, 77].

2.2 Authoring System

2.2.1 System using High-level Input

There have been many previous systems that generate animations from high-level descriptions. Perlin's *Improv* system [88] creates actors that respond to users and to each other in realtime, using procedural animation and a rule-based behavior engine that takes as input English-style script describing the actors' communication and decision-making skills. Similar AI approaches have been proposed to generate group behaviours during social interactions and other scenarios: Pedica et al. [83] model the territorial dynamics of social interactions; Yu and Terzopoulos [122] simulate pedestrians in urban settings using hierarchical decision networks, while Funge et al. [22] introduce the cognitive modeling language CML, to endow their autonomous characters with enough intelligence to work out a detailed sequence of actions to satisfy a brief behavior outline or "sketch plan". Calvert and Ma's [10] *Life Forms* system

presents a selection of keyframed motion sequences from a large database based on a choreographer's sketch of a dance motion, and provides support for further editing using procedural animation and inverse kinematics. The authored dance sequences can then be viewed on multiple virtual characters. We derive inspiration from this approach in the design of user interface for our complete scene generation system.

Another related body of work involves the automatic generation of conversational motions. The Behavior Expression Animation Toolkit (BEAT) takes as input a typed script and generates appropriate and synchronized nonverbal behaviors and synthesized speech [11]. Stone et al. [103], and later Levine et al. [62], use a database of recorded speech (composed of short, clearly-delimited phrases), while Neff et al. [82] analyzes video to create a statistical model of a specific performer's gesturing motions, and then generates full-body animation in the style of that individual for any novel input text.

2.2.2 User-interactive System

Several systems focused more on user interaction when generating motions by providing a wide variety of candidate scenes, from which they can intuitively select and refine. Our complete scene generation system have also incorporated design elements from several relevant systems. Marks et al. [71] introduced the concept of *Design Galleries*, which present the user with a broad selection of perceptually different graphics or animations by varying sets of parameters. The main criteria for their approach are dispersion (i.e., sampling the full set of possible outputs) and arrangement (i.e., providing an intuitive browsing and selection interface). Similarly, *Many-Worlds Browsing* exploits the speed of multibody simulators to compute numerous example simulations, and allows the user to browse and refine them interactively [108], while *Physics Storyboards* have been proposed to focus on key events and outcomes to accelerate the process of tuning interactive, procedural animations [30].

2.3 Shadow Theatre

2.3.1 Shadow Generation

Shadows have long been a key research topic in the computer graphics community because they lend realism and hint at spatial relationships among objects. Many researchers have extensively investigated the rapid and realistic generation of shadows. Crow et al. [17] proposed the shadow volume technique, which synthesizes shadows by computing cones that enclose a light source and objects. Williams et al. [115] introduced shadow mapping, which utilizes both pre-rendered depth information obtained from a light source view and a transformation from the light source to the original camera, with this result later used to determine areas of occlusion. These are for local illumination models and the current de facto techniques in real-time graphics applications. For global illumination methods (e.g., ray-tracing, photon mapping, or radiosity), an additional process is not required since its illumination model already reflects the shadow effect [5, 114, 25, 40].

2.3.2 Shadow for Artistic Purpose

Shadows for artistic purpose have also been studied. Pellacini et al. [84] created an interface for placing shadows as user requirements, where the lights or objects are replaced accordingly. Kry et al. [48] demonstrated an animated hand based shadow puppets by using glove sensors and their real-time skinning technique. Mitra et al. [74] proposed a tool for sculpting 3D geometries with meaningful shapes when projected onto several non-orthogonal planes, while Bermano et al. [7] exploited self-shadowing effects to put multiple images into one 3D printed picture, which can be viewed when different light sources are turned on. Mattausch et al. [72] directly manipulated rendered shadows and applied the edited results in subsequent scene rendering.

2.3.3 Viewing Shadow Theatre as Collages/Mosaics of People

The methods for synthesizing collages or mosaics are also relevant from the perspective that the resultant shadow is a collage of shadows made by human bodies. Hausner et al. [33] created a decorative mosaicking algorithm by placing tiles along user-selected edge features based on a direction field. Collages in non-image domains have also been studied. Gal et al. [23] modeled expressive 3D characters using primitive 3D models. Kim et al. [45] synthesized crowd scenes by tiling motion patches that are primitive motion units in the scenes.

2.4 Physics-based Controller Design

2.4.1 Controllers for Various Characters

Many researchers have explored a variety of motion control methodologies for simulated virtual characters in the past few decades. Achieving visual realism, responsiveness to interaction, robustness to external perturbation, and adaptability to condition change are the design goals of physically based simulation and control. Although biped control have gained the most attention [102, 120, 58, 49, 112, 59, 29, 66], nonhuman creatures such as quadrupeds [16], swimming fishes [26, 105], flying birds [117, 42], and imaginary deformables [15, 104] have also been studied.

2.4.2 Trajectory Optimization

Many optimal control methods focus mainly on generating a single trajectory that minimizes a certain energy function. The key challenge of trajectory optimization (a.k.a. spacetime optimization) is the handling of discontinuous events in the trajectory. Derivative-free, sampling-based optimization methods, such as downhill simplex and CMA-ES, are proven to be quite effective for trajectory optimization [102, 105, 3, 29]. Alternatively, contact-invariant optimization leverages fictional force at contact points to yield smooth energy landscape, for which fast, derivative-based optimization methods can be exploited [78]. Energy-optimal motion control is often fragile at

the presence of unexpected perturbation because the effort for control is minimized. Wang et al [111] leverages stochastic optimal control to improve the robustness of the optimized trajectory at extra energy cost.

2.4.3 Sampling-based Optimization

Generating an open-loop simulation requires the optimizer to work directly on torque profiles at actuated joints [102, 67] or a mechanism/controller that produces torques [117, 109, 105, 68]. Small changes in actuated torques may result in large changes in the simulated trajectory and therefore the optimization space is not continuous. Newton-type methods would fail at the presence of a discontinuity, because first- and second-order derivatives do not provide reliable information about the optimization process. Instead, sampling-based optimization techniques have frequently been employed for optimizing open-loop simulations [109, 111, 112]. The key idea is to distribute random samples to decide the next direction to move in the iterative loop of the optimization process. A stochastic decision based on random samples is usually more robust than choosing a direction based on first- and second-order derivatives. Sampling-based methods, such as Covariance Matrix Adaptation (CMA), are more robust and less prone to fall into local minima, but less efficient than newton-type methods [31].

2.4.4 Model-Based Controller Design

Many locomotion controllers have an underlying control model that drives the dynamics system to move on. The most popular model is a phase-based state machine, which has keyframes at state nodes and rules for phase transitioning [120]. Data-driven control leverages motion capture to improve visual realism and provide fine-level control over locomotion [58]. Alternatively, simulated controllers can be designed based on the principle of least effort, seeking optimal control policies that minimize either total joint torques or metabolic energy expenditure [111, 112, 59].

2.4.5 Direct Policy Learning

The controller can be viewed as a direct mapping π from state s to action a such that $a = \pi(s)$. Such a direct policy can be constructed from a collection of state-action observations (s_i, a_i) via regression. Sok et al [102] collected state-action observations from the motion capture of human locomotion. Simply tracking motion capture data does not reproduce the original locomotion because the data may include modeling and acquisition errors. They employed trajectory optimization to rectify motion capture data and then applied locally-weighted linear regression to construct a control policy. Levine and his colleagues [60, 61] presented guided policy search methods that combine the flexibility of trajectory optimization and the generality of function approximation by solving them alternately. Their control policy is represented as neural networks. Trajectory optimization based on iterative LQG generates a collection of long-term, high-quality training data for neural network learning, and the learned policy thus obtained provides a new set of trajectories to be optimized for the next iteration. They applied their guided policy search methods to end-to-end visuomotor robot arm manipulation tasks and learned policies that maps raw image observations directly to torques at the robot's motors. Mordatch and Todorov [76] presented a similar method that uses the Alternating Direction Method of Multipliers (ADMM) to couple the neural network and trajectory optimization. Tan et al [106] developed a control system for controlling a human character to ride a bicycle in a physically simulated environment. They employed CMA-based trajectory optimization to design feedforward controllers for simulating short-term energetic motions, while a neural network evolution method is used to learn direct control policies for simulating balance-driven motions.

2.4.6 Deep Reinforcement Learning for Control

Deep reinforcement learning employs deep neural networks as function approximators. Human-level Atari game play has been achieved by using state-action value approximators of each discrete action and replay buffers [75, 93]. Having both state-

value and policy functions as deep neural networks allows the policy to be updated by computing analytic gradients (i.e. policy gradient). Schulman and his colleagues [95, 94] showed that choosing step-size carefully for policy gradient methods is crucial to ensure stable and steady improvements of the policy. Silver and his colleagues [99, 64, 34] demonstrated standard RL benchmarks for continuous control (e.g. cart pole balancing, arm swing-up, arm reaching) using DRL. Peng and his colleagues [85] learned terrain-adaptive locomotion skills of planar bipeds and quadrupeds using their dynamic states and terrain descriptions as input and parameterized steps and jumps as output actions. They also demonstrated 3D biped locomotion controller that can navigate along the target by using hierarchical deep reinforcement learning framework, where the low-level controller deals with the balance and style while the high-level controller deals with high-level goals [87].

Chapter 3

Motion Generation for Shadow Theatre

3.1 Overview

Shadow theatre is a genre of performance art that delivers stories in a visual form. What makes it different form other genres is that it utilizes shadows as the only medium of communication. Because audiences can only see the shadows created by the actors on a screen, this stimulates their imaginations regarding what is happening behind the scene. As a result, the audience is immersed in the performance. Although puppets and human hands have been the primary objects used to compose these shadows, several performance teams have created their work using the whole bodies over recent years. Modern lighting technologies have enabled larger stage areas to be exploited for full-body shadow plays, which may raise the level of immersion.

The aim is to reproduce scenes from shadow theatre performances in an automatic manner. Given a sequence of shadow images or silhouettes, we would like to generate the motion of animated 3D characters that cast shadows matching the target. The settings used in state-of-the-art performances, where multiple actors and environments such as props and platforms come on the scene, are also considered. Using

the proposed system, the user can explore plausible and creative poses not only for the shapes already used in modern performances but also for new shapes that have not yet been exploited. The system can also be used to model articulated sculptures for the purpose of shadow art.

The basic components that generate shadows during a performance are the light source, the screen, and the actors. If the relative positions and orientations are selected for the components, the shadows on the screen are determined accordingly. The light and screen are usually fixed, whereas the actors move around the stage. Therefore, coordination between the actors and their poses in the spatio-temporal domain are key elements in constructing the scene. However, there are several challenges to be addressed. The first challenge is that certain target shapes require multiple actors to pose collaboratively while coordinating their shadows carefully. The second challenge is the completeness of the resulting shadows, i.e., not only should the outlines of the shadows match the target shape but the internal area should also be filled completely. The third challenge is to ensure spatio-temporal coherence of the actors' motions such that smooth target animations can be created. The last challenge is incorporating the nontrivial constraints existing in actual performances (e.g., collisions between actors, physical plausibility of the resultant motions, limited stage area).

The problem is high-dimensional and inherently ill-posed. Our approach is to formulate it as a nonlinear constrained optimization problem with objectives. The formulation of the objectives was inspired by how professional actors train themselves and perform on stage. We found that they have heuristic strategies, which facilitate our optimization effectively and thus make it feasible. We will demonstrate the effectiveness of our approach with a variety of target silhouettes and animations. The results are further validated by means of 3D fabrication.

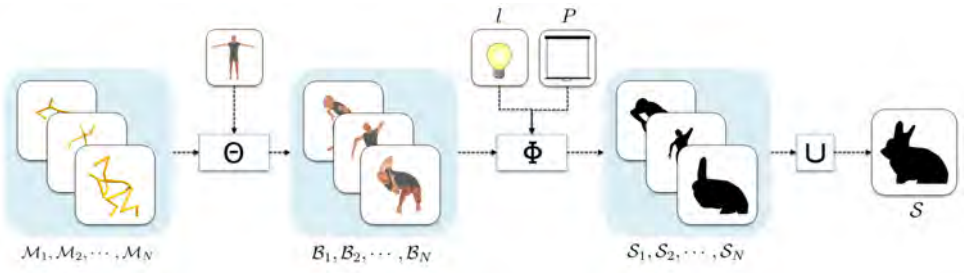


Figure 3.1 The overall process of shadow generation. Skeletal motions $\mathcal{M}_1(t), \mathcal{M}_2(t), \cdots, \mathcal{M}_N(t)$, their neutral body meshes and skinning method Θ generate animated body meshes $\mathcal{B}_1(t), \mathcal{B}_2(t), \cdots, \mathcal{B}_N(t)$. Given a stage with a light source l and a screen P, their shadows $\mathcal{S}_1(t), \mathcal{S}_2(t), \cdots, \mathcal{S}_N(t)$ are generated and the final result on the screen is shown as a sum of those shadows.

3.2 Shadow Theatre Problem

3.2.1 Problem Definition

A scene in shadow theatre is composed of a set $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \cdots \cup \mathcal{S}_N$ of N actors' shadows on screen plane $P = (\mathbf{n}, d)$, where $\mathbf{n} \in \mathbb{R}^3$ and $d \in \mathbb{R}$ are the normal vector and a constant respectively, of the plane (see Figure 3.1). Each actor's shadow $\mathcal{S}_n$ is determined by its original body shape $\mathcal{B}_n$ in the 3D space and the projection function $\Phi : \mathcal{B}_n \to \mathcal{S}_n$, which is defined using a point light source $l \in \mathbb{R}^3$. The body shape $\mathcal{B}_n$ is derived from a 3D skeletal posture $\mathcal{M}_n = (\mathbf{v}_0, \mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_M)$ and a mesh skinning function $\Theta : \mathcal{M}_n \to \mathcal{B}_n$, where $\mathbf{v}_0 \in \mathbb{R}^3$ and $\mathbf{q}_0 \in \mathbb{S}^3$ are the position and orientation of the root joint respectively, $\mathbf{q}_m \in \mathbb{S}^3$ for $m > 0$ is the relative orientation of joint m with respect to its parent joint (i.e., joint space representation), and Θ is a linear skinning model. The input to our system is a target shadow $\mathcal{T}$, which is represented as a binary image. The output of the system is a set of N actors' full-body poses $\{\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_N\}$ that projects shadows $\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_N$ on the screen matching the target shadow. The problem generalizes easily to take a sequence of target shapes (2D shadow animation) as input and produce a coordinated animation of N actors (3D animation of full-body characters).

Figure 3.2 Experiments with professional actors. We provided target shapes and recorded how they pose to cast a matching shadow. Here, two actors initially shaped an outline of a triangle and another one crouching at the front filled the remaining holes inside.

3.2.2 Approaches of Professional Actors

Discovering 3D full-body poses from a 2D target shape is an inherently ill-posed problem. Many different sets of poses can generate similar shapes when projected onto a screen. The number of unknown variables to be determined is large (N for actors $\times$ M for joints $\times$ T for time frames); the sum exceeds one thousand, even for a few seconds of performance. Moreover, the process from $\mathcal{M}$ to $\mathcal{S}$ is highly non-linear and non-intuitive because it is a composition of kinematics, skinning, and projection functions. Therefore, naive approaches such as trial and error or exhaustive search are almost impractical. In order to manage these difficulties, we recorded the training session of professional actors to understand how they pose to match the target shapes we provided (see Figure 3.2). Through the training session, we learned that they have heuristic yet effective strategies, as summarized below.

Strategy 1 *(Principal poses)*. Given an animation of target shapes, they first determine a representative shape at a time frame, for which they pose to cast shadows. The poses that generate the representative shape are called *principal poses*. The full character animation is generated by adapting principal poses gradually over the other frames.

Strategy 2 *(Distinctive features)*. The actors have extensive prior knowledge about which body parts match particular features (e.g., a snout and ears for a rabbit, a trunk and ears for an elephant) in the target shape. For example, they tend to use elbows and knees to match sharp angles and vertebral or gluteal-femoral areas to match round curves.

Strategy 3 *(View direction)*. The actors should be capable of looking at the screen while posing in performances. Since shadow performance requires delicate body control and adjustment continuously, the constraint on viewing direction is crucial.

Strategy 4 *(Balance)*. Balance and stability are also important issues. If several poses are equally plausible for a target shadow, more stable poses are preferred.

Strategy 5 *(Scaling)*. Although the shadow cast by a point light can be scaled easily by moving the actors between the light and the screen, the size of the target shadow matters if the stage area is limited. Actors use a simple rule stating that fewer should join for smaller shapes and more should join for larger shapes.

Strategy 6 *(Minimal motion)*. Once principal poses at the representative frame are determined, the actors adjust their poses to track a sequence of animated target shapes continuously. While doing so, they tend to minimize joint movements and deviation from the principal poses. They typically do not change their ground contact states while tracking a continuous sequence in order to maintain their balance and stability.

3.3 Discovery of Principal Poses

3.3.1 Optimization Formulation

Given a 2D target shape, we formulate the discovery of principal poses as non-linear optimization of an energy function, which is designed to reflect the heuristic strategies and stage constraints in performances. The principal poses of N characters are computed simultaneously in a single energy optimization.

$$\begin{aligned} \underset{\mathcal{M}_1,\mathcal{M}_2,\cdots,\mathcal{M}_N}{\operatorname{argmin}} \quad & E_{\text{contour}} + E_{\text{coverage}} + E_{\text{coherence}} + \\ & E_{\text{visible}} + E_{\text{physics}} + E_{\text{hint}} \end{aligned} \tag{3.1}$$

$$\text{subject to} \;\; \mathbf{a_l} \leq \Gamma(\mathcal{M}_n) \leq \mathbf{a_h} \;\; \text{for} \;\; n = 1, 2, ..., N.$$

Here, $\{\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_N\}$ denotes a set of full-body poses, Γ is a joint angle measure function. $\mathbf{a_h}$ and $\mathbf{a_l}$ are the upper and lower bounds for all joints, respectively. We employed axial, conic, and spherical constraints in the unit quaternion space to design the joint measure function [50]. The first term E_{contour} penalizes visual differences between the resultant and target shadow contours on the projection screen.

$$\begin{aligned} E_{\text{contour}} = w_1 \sum_i \|\mathbf{p}_i - \mathbf{p}'_i\|^2 + w_2 \sum_i \|\mathbf{n}_i - \mathbf{n}'_i\|^2 + \\ w_3 \sum_i \|\kappa_i - \kappa'_i\|^2 \end{aligned} \tag{3.2}$$

where $\mathbf{p}_i$ is a point sampled on the contour of the target shape and $\mathbf{p}'_i$ is its closest point on the resultant shadow contour. $\mathbf{n}_i$ and κ_i are the normal and curvature at point $\mathbf{p}_i$, respectively.

The second term E_{coverage} measures the area difference between the resultant and the target shape (see Figure 3.3).

$$E_{\text{coverage}} = w_4(\text{Area}(\mathcal{T} \setminus \mathcal{S}) + \text{Area}(\mathcal{S} \setminus \mathcal{T})) \tag{3.3}$$

where $\mathcal{S}$ is the shadow of the characters, $\mathcal{T}$ is the target shape, and $\setminus$ represents the area difference operator on the 2D geometry. This term is a supplement to the

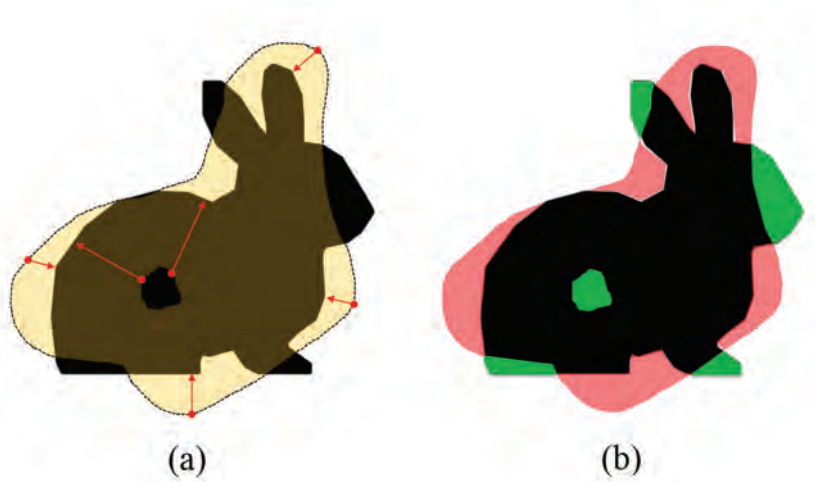


Figure 3.3 Contour and coverage differences. (a) The contour-related terms are computed between points sampled on one contour and their closest points on the other contour. (b) The coverage terms minimize the mutual subtraction of the shape areas. $(\mathcal{S} \setminus \mathcal{T})$ and $(\mathcal{T} \setminus \mathcal{S})$ are shown in red and green, respectively.

contour difference for shape matching, and it also prevents unexpected holes inside the contour.

It is preferred in shadow performances for the shadow of each individual actor to adhere to a coherent sub-section of the target contour, as being responsible for multiple, scattered sections is burdensome. The third term $E_{\text{coherence}}$ encourages the contour to be divided into a small number of pieces such that the shadow of each individual character corresponds to a few continuous pieces,

$$E_{\text{coherence}} = w_5 \sum_i \text{CountIf}(\text{Whose}(\mathbf{p}_i) \neq \text{Whose}(\mathbf{p}_{i+1})) \tag{3.4}$$

where $\text{Whose}(\mathbf{p}_i)$ provides the index of an actor whose shadow is closest to point $\mathbf{p}_i$. CountIf returns one if the argument is true, and zero otherwise.

The fourth term E_{visible} favors poses with their viewing direction towards the

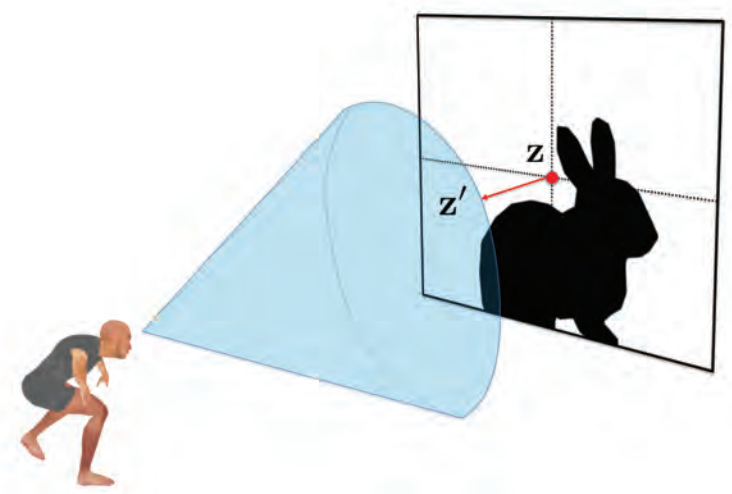


Figure 3.4 Visibility: The optimization minimizes the distance from the center of the screen to the viewing cone.

projection screen,

$$E_{\text{visible}} = \begin{cases} w_6 \sum_n \|\mathbf{z} - \mathbf{z}'_n\|^2, & \text{if the cone reaches the screen} \\ \infty, & \text{otherwise} \end{cases} \tag{3.5}$$

where $\mathbf{z}$ is the center of the screen and $\mathbf{z}'_n$ is the closest point on a viewing cone that approximates the visible volume of the n-th actor (see Figure 3.4).

The fifth term E_{physics} favors poses that are physically plausible. Each character should be in contact with the floor, balanced, and robust against external perturbations.

$$\begin{aligned} E_{\text{physics}} = w_7 \sum_n \text{DistFloor}(\mathcal{B}_n) + \\ w_8 \sum_n \text{Area}(SP_n) + \\ w_9 \sum_n \text{Margin}(SP_n, COM_n) + \\ w_{10} \sum_n \sum_m \text{PtDepth}(\mathcal{B}_n, \mathcal{B}_m) + \\ w_{11} \sum_n \sum_k \text{PtDepth}(\mathcal{B}_n, \mathcal{E}_k) \end{aligned} \tag{3.6}$$

where DistFloor computes the minimum distance between the body of an actor and the floor. SP_n and COM_n are the support polygon and center of mass projected on the ground of the n-th actor, respectively. The support margin (Margin), which is the signed distance from COM_n to the boundary of SP_n, measures the robustness of a statically-balanced posture. Additional terms are used to deal with person-to-person and person-to-environment (e.g., floor, screen and props) collisions. $\mathcal{B}_n$ is a skinned body mesh and $\mathcal{E}_k$ is the geometry of an environment object. The penetration depth (PtDepth) measures the degree of interpenetration between objects. Note that PtDepth also measures the self-penetration depth if the same object is given.

E_{hint} is a term which allows the user directly to interfere with the optimization process. The user can exploit heuristic knowledge with this term, as described in *Strategy 2*,

$$E_{\text{hint}} = w_{12} \sum_i \min(\|\mathbf{h}_i - \mathbf{h}'_{i1}\|^2, ..., \|\mathbf{h}_i - \mathbf{h}'_{ik}\|^2) \tag{3.7}$$

where $\mathbf{h}_i$ is a hint point on the target contour and $\mathbf{h}'_{ik}$ is a matching point on the character's body. For example, $\mathbf{h}_i$ is an ear tip of a bunny shadow, and we may want to use a character's elbow to match the tip. Then, $\{\mathbf{h}'_{i1}, \cdots, \mathbf{h}'_{ik}\}$ includes the right and left elbows of all characters on the stage. E_{hint} attracts the closest elbow to the hint point.

3.3.2 Optimization Algorithm

The CMA-ES [31] was adopted to solve our optimization problem. Although CMA-ES was shown to be a powerful nonlinear solver in many previous studies, it often converges to sub-optimal solutions for our problem due to its high-dimensionality and nonlinearity. Sub-optimal solutions can cause visual artifacts, such as holes in shadows and contour mismatches. Our optimization algorithm takes CMA-ES as a basis and adds extra steps to escape from local extrema. If CMA-ES converges to a solution and the residual energy in equation 3.1 is above a user-specified threshold or visible artifacts are present in the solution, the extra steps are activated to search for better solutions based on two key ideas (see Figure 3.5).

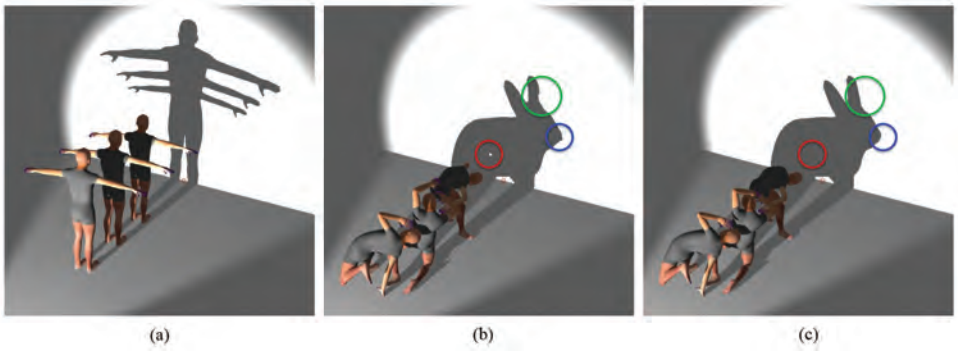


Figure 3.5 Refinement of principal poses. (a) All actors initially T-pose in a row. (b) CMA-ES generates poses with artifacts such as holes and silhouette mismatches. The foremost actor's left arm was completely occluded by the other body parts, so does not contribute forming the shadow. (c) The local refinement steps moved the occluded arm to fill in the hole and matched the shadow contour more precisely.

The first idea is to exploit body parts that are completely occluded by other body parts, other actors, or props. Such occluded body parts do not contribute to forming shadow contours or filling holes can thus be manipulated freely. Starting from individual end-effectors, we check if the body part is completely occluded and move on to its parent link to repeat. In this way, we can identify a chain of completely occluded body parts sequentially. Re-running the CMA-ES algorithm with only the occluded body chain, while leaving the remaining parts fixed, may refine the solution. If there is no such chain, the second idea is to reduce the dimensionality of the problem by selectively including degrees of freedom that may contribute to removing visual artifacts immediately. To do so, the body parts, of which shadows are contiguous to holes or contour mismatches, are selected for the refinement. CMA-ES with reduced degrees of freedom often converges to better solutions. If the refinements are still unsatisfactory, the last resort is to add a new actor to the scene. We add actors one-by-one and repeat the whole process until the resultant principal poses are satisfactory.

3.4 Animating Principal Poses

In this section, we discuss how to deal with a series of target shadows to generate character animation. Our algorithm follows the strategies of professional actors; we first select a representative frame for which the algorithm from the previous section generates principal poses. The principal poses serve as an initial guess for optimizing the poses at neighboring frames. This optimization process propagates to neighboring frames until a full animation is generated. Without loss of generality, we assume that the representative frame is t_0 and the optimization propagates forward to compute poses at frames t_i for $i > 0$.

3.4.1 Initial Configuration

Providing the optimization solver with a good initial guess is important, particularly when the problem is high-dimensional and nonlinear. We assume that the interiors of the target shapes are triangulated into K pieces consistently (see Algorithm 1). Let $\mathcal{T}^i = \mathcal{T}(t_i)$ be a target shape at frame t_i with its interior triangulated. Let $\mathcal{M}_n^i$ be a pose of the n-th character at frame t_i and $\hat{\mathcal{M}}_n^i$ be a long vector concatenating all joint positions in the reference system. Applying the forward kinematics map to $\mathcal{M}_n^i$ produces $\hat{\mathcal{M}}_n^i$. The light source and the target shape at frame t_i forms a generalized cone $\mathcal{G}^i = \mathcal{G}_1^i \cup \cdots \cup \mathcal{G}_K^i$ of K tetrahedrons, where each tetrahedron is composed of the light source and one of the interior triangles in $\mathcal{T}^i$ (see Figure 3.6). Then, any joint of a character is positioned in a tetrahedron, and its barycentric coordinates in the tetrahedron specify the joint position in the generalized cone $\mathcal{G}^i$. Let C_n^0 be the barycentric coordinates of all joint positions of principal poses $\hat{\mathcal{M}}_n^0$ with respect to generalized cone $\mathcal{G}^0$ (line 2–3).

For any $\mathcal{T}^i$, we would like to estimate the character's poses roughly with the initial barycentric coordinates C_n^0 with respect to generalized cone $\mathcal{G}^i$ at frame i (line 7). Any joint at frame t_0 has a tetrahedron $\mathcal{G}_k^0$ containing the joint and its barycentric coordinates. This joint is mapped to a new position at frame t_i, which is determined

using the barycentric coordinates in $\mathcal{G}_k^i$. Given the joint estimates, determining the poses at a new frame can be formulated as an inverse kinematics problem (line 8). We used a standard technique based on damped Jacobian pseudo inverse and line minimization to solve for inverse kinematics.

Algorithm 1 Adapting principal poses for new target shapes

t_0 : representative frame

$\mathcal{T}^i$: target shape at frame t_i

$\mathcal{G}^i$: generalized cone at frame t_i

$\mathcal{M}_n^i$: the pose (joint configuration) of n-th character

$\hat{\mathcal{M}}_n^i$: joint positions of n-th character

1: **for** $n \leftarrow 1, \cdots, N$ **do**

2: $\quad \hat{\mathcal{M}}_n^0 \leftarrow \text{ForwardKinematics}\big(\mathcal{M}_n^0\big)$

3: $\quad C_n^0 \leftarrow \text{BarycentricCoord}\big(\mathcal{G}^0, \hat{\mathcal{M}}_n^0\big)$

4: **end for**

5: **for** $i \leftarrow 1, \cdots, T$ **do**

6: $\quad$ **for** $n \leftarrow 1, \cdots, N$ **do**

7: $\quad\quad \hat{\mathcal{M}}_n^i \leftarrow \text{BarycentricCoord}^{-1}\big(\mathcal{G}^i, C_n^0\big)$

8: $\quad\quad \mathcal{M}_n^i \leftarrow \text{InverseKinematics}\big(\hat{\mathcal{M}}_n^i\big)$

9: $\quad$ **end for**

10: $\quad \mathcal{M}^i \leftarrow \text{Optimize}\big(\mathcal{T}^i, \{\mathcal{M}_n^i\}\big)$

11: **end for**

3.4.2 Optimization for Motion Generation

Once the poses at frame i are estimated, the next step is to run optimization with those poses as the initial configuration (line 10). The optimization process for motion generation is similar to the one in the previous section except that three additional energy terms (E_{smooth}, E_{regul}, $E_{contact}$) are exploited. E_{smooth} ensures temporal co-

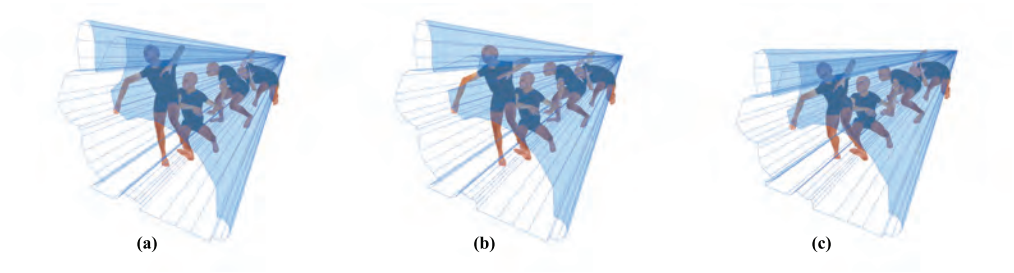


Figure 3.6 Motion generation. (a) The principal poses at the representative frame. The rays from the light source form a generalized cone with its base matching the target silhouette. (b–c) adapting the principle poses to similar target shapes. The poses change smoothly and coherently across frames.

herence between frames preventing jerky animation,

$$E_{smooth} = w_{13} \sum_{n} \text{Diff}(\mathcal{M}_n(t-1), \mathcal{M}_n(t)), \tag{3.8}$$

where the pose difference at two frames measures the discrepancy of the root positions and the joint angles.

$$\begin{aligned} \text{Diff}(\mathcal{M}(t_1), \mathcal{M}(t_2)) = w^v \|v_0(t_1) - v_0(t_2)\|^2 + \\ \sum_m w_m^q \|q_m(t_1)^{-1} q_m(t_2)\|^2. \end{aligned} \tag{3.9}$$

Here, $\|q_m(t_1)^{-1} q_m(t_2)\|$ is the geodesic distance between unit quaternions $q_m(t_1)$ and $q_m(t_2)$. w^v and w_m^q weigh the significance of individual degrees of freedom.

The regularization term E_{regul} prevents excessive deviations from initial principal poses,

$$E_{regul} = w_{14} \sum_{n} \text{Diff}(\mathcal{M}_n(t_0), \mathcal{M}_n(t)) \tag{3.10}$$

where t_0 is the index of the representative frame where the principal poses are computed.

The contact term $E_{contact}$ prevents contact changes in character animation based on *Strategy 6* thus, the initial contact states at the representative frame remain un-

		Expected trials	Time per trial	Total time
Simple Shapes	Triangle	1.25	3.48	4.35
	Rectangle	1.25	3.47	4.34
	Circle	2.5	3.65	9.13
Complex Shapes	Elephant	25	6.69	167.25
	Hat	3.33	5.82	19.38
	Highheel	12.5	6.6	82.5
	Mountain	8.44	6.87	57.98
	Rabbit	33.33	6.98	232.64
	Tshirt	23	7.37	169.51
	Car	12.5	6.51	81.38
	Airplane	2.5	5.93	14.83
	Violin	3.33	5.84	19.45

		Key frames	Time per frame	Total time
Animations	Rabbit	16	4.5	72
	Elephant	34	3.66	124.44
	Car	12	3.86	46.32

Table 3.1 Runtime performance. The computation time is measured in minutes.

changed throughout the optimization process.

$$E_{contact} = \sum_{j \in \text{contact}} w_{15} \| c_j^v(t_0) - c_j^v(t) \|^2 + w_{16} \| c_j^q(t_0)^{-1} c_j^q(t) \|^2 \tag{3.11}$$

where c_j^v and c_j^q are the position and orientation, respectively, of the j-th body part with respect to the reference system, if the body part is in contact with the ground surface.

3.5 Experimental Results

Since CMA-ES is based on stochastic sampling, optimization would converge to different solutions with different random seeds even if the algorithm begins with the same initial configuration. Therefore, we select the best result among multiple optimization trials. The population size of CMA-ES is $\lambda = 40$. The optimization at each trial converges within one thousand iterations for simple shapes and within two thousand iterations for complex shapes.

The algorithm runs on a desktop PC equipped with an Intel Core i7 4790K (4 cores, 4.0GHz) and NVIDIA GeForce GTX 480. Table 3.1 presents the performance statistics. The table shows expected trials until we get one satisfactory result. If the expected trials is 3, one out of three results is satisfactory. The notion of satisfaction is admittedly subjective and dependent on our artistic sense. Our criteria include two measures. First, the result should be recognizable as was intended. For example, an elephant's silhouette should be recognizable as an elephant for any viewer. Second, the result should reproduce the key features of the target silhouette. To discover the principal poses, our algorithm can find plausible solutions for most of our examples within a few trials and 5 to 20 minutes of computation except for several challenging examples. The rabbit in Figure 3.8 is the most difficult, requiring dozens of trials until a satisfactory result is obtained and 4 hours of computation time. Although the time complexity is basically proportional to the number of characters and the number of key features in the target shape, there are other factors that may affect the computation time significantly. For example, the inverted triangle has a wide horizontal span at the top and narrow support at the bottom, which makes it difficult to find balanced stable poses. The silhouette of the t-shirt in Figure 3.8 is simpler than those of the elephant and the car, but the t-shirt requires more computation time since it has narrow support, which makes the characters pose acrobatically. To generate animations, only one trial is enough for each key frame because we start the adaptation process with plausible initial poses. It takes 50 to 125 minutes to generate entire animations, and computation time for a single optimization is reduced as compared to discovering principal poses due to its fast convergence.

3.5.1 Implementation Details.

The implementation of the system includes many technical components and acceleration techniques. An open source game engine [**Ogre**] was used to implement character skinning and shadow rendering. The built-in shader functionality of the engine was modified to color-code the pixels in the shadows by character identifiers. In order to

compute the penetration depth efficiently, we approximate the 3D character's body volume using spheres. The $E_{contact}$ term requires CMA-ES to solve inverse kinematics as a part of a big stochastic optimization. We decouple inverse kinematics from the optimization and run a standard inverse kinematics solver for every CMA-ES sample to manage the contact term separately.

3.5.2 Animation

Beginning with a 2D silhouette, we used a shape manipulation technique [38] to generate 2D shadow animation (see Figure 3.10 and the supplemental video). For the rabbit and elephant examples, the target animations were designed to have large movements of their perceptual features. The rabbit moved its ears back and forth while lowering its head. The elephant raised its trunk as if trumpeting. For the car example, we choreographed a scene with a person driving a car, which is jerking up and down.

3.5.3 3D Fabrication

We 3D-printed principle poses using an *Object 24* printer (see Figure 3.9). The mesh of a resultant pose requires post-processing steps to be fed into the 3D printer. We remove the self-collision of the mesh, which is an artifact of linear skinning deformation. We also reinforce fragile support between the character's feet and the ground. Two lighting devices were tested for stage installation. The first device was a *Cree* LED light module, which is small yet bright to imitate an ideal point light source. It can also be used as a spot light if a convex lens is installed. The second device is a portable projector *Optoma ML750*, which emulates an actual stage setting because professional actors use a large-scale projector as their light source in stage performance. The fabrication results consistently generated shadows similar to the target shapes with both lighting devices.

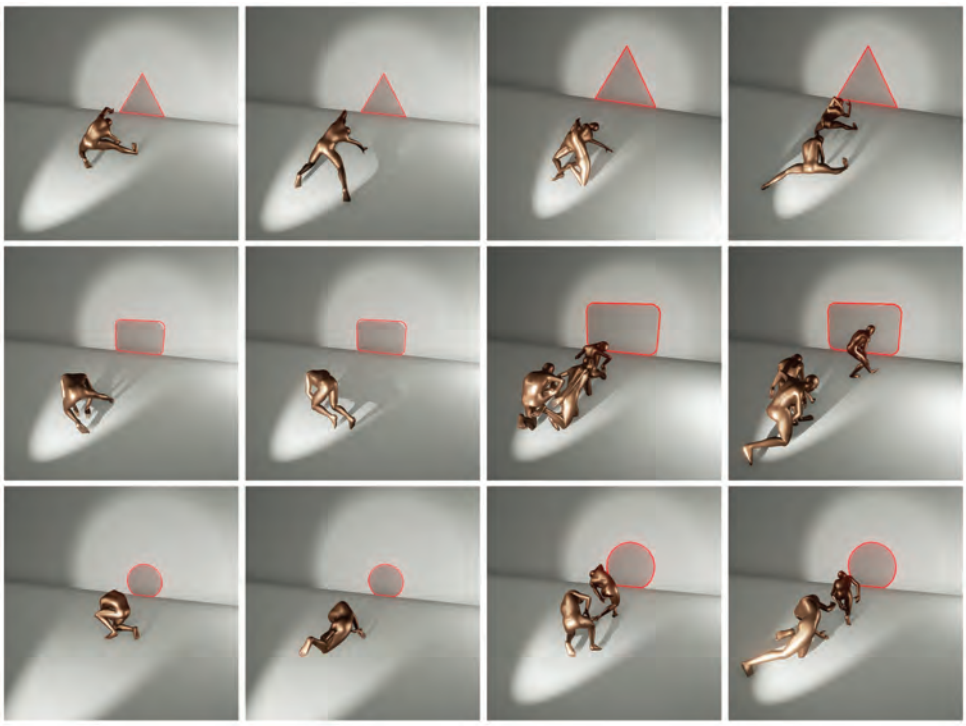


Figure 3.7 Principal poses for simple shapes. (1st column) principal poses for target shapes shown in red line. (2nd column) Alternative solutions at another local minima. (3rd and 4th columns) The target shapes are scaled by a factor of 1.5 and thus require more actors to join in.

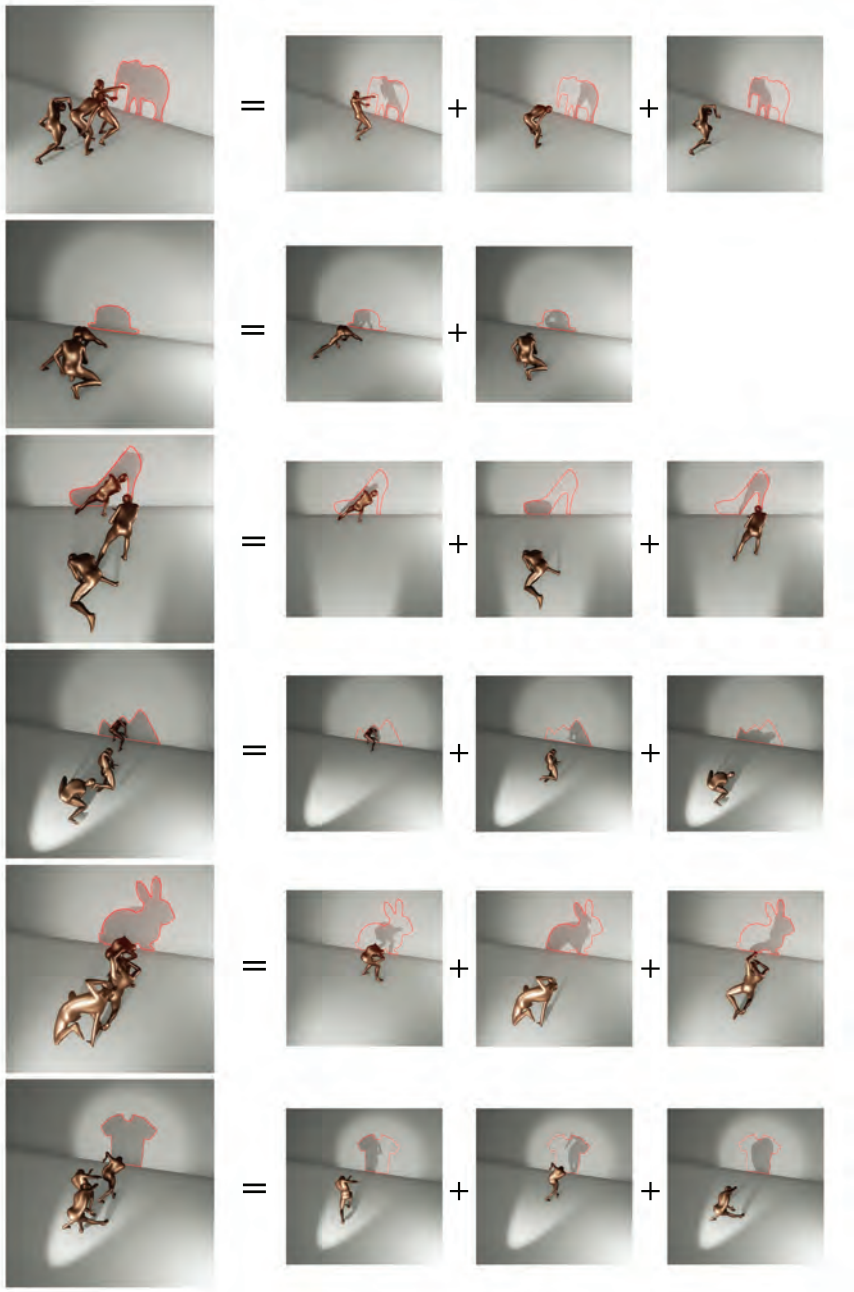


Figure 3.8 Principal poses for complex shapes. In each row, the leftmost image shows the optimized poses while the images on the right show the shadows of individual actors.

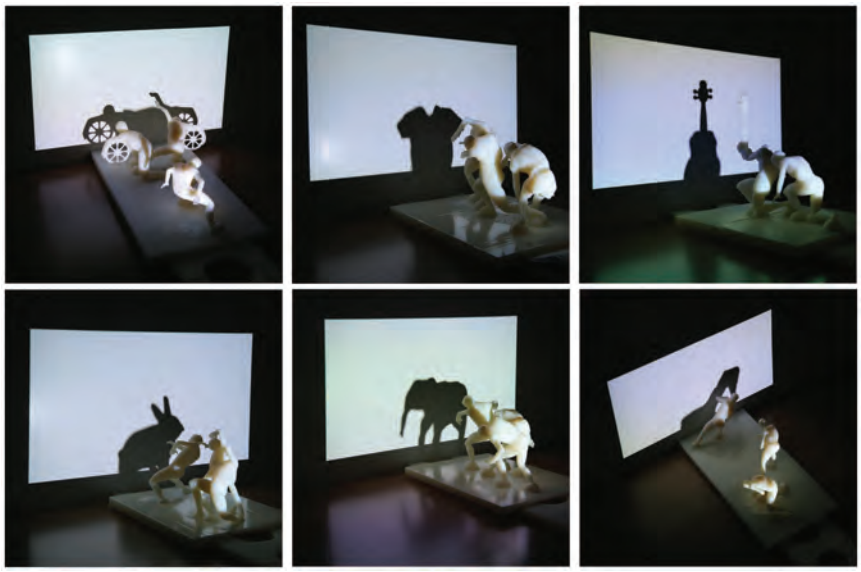


Figure 3.9 3D Fabrication results.

3.6 Discussion

The approach is a large optimization problem, consisting of many non-linear components such as kinematics, kinetics, skinning, perspective projection, and multi-character coordination. The biggest challenge is to make the optimization converge to a plausible solution with reasonable computing resources. The key to success was the design of the objective functions, which compete with each other and thus keep the optimization process balance on plausible sub-manifolds in the vast search spaces. The heuristic strategies learned from professional actors also narrowed down the search space substantially and made the results human-like. Selective refinement of DOFs effectively localized the large optimization to reduce it into a series of smaller optimization tasks.

Our optimization problem has tens of weight parameters that should be tunned to make it work in practice, which could be burdensome for novice users. Our experience for tuning the parameters is as follows. First, initial weights were set so that all energy terms are as equal as possible. This could be done by measuring ranges of values of

the terms. Second, we distinguished critical terms and modified them first. In our case, the balance between contour, coverage and physics terms were critical. Finally, we added remaining terms one by one. Whenever a new term was added, its weight value was increased until the relevant artifacts were removed.

There are a number of limitations in this approach. Artifacts of linear skinning and inaccurate body modeling affect shadow generation particularly when the characters pose acrobatically. More realistic body models based on either biomechanical or data-driven modeling would improve the quality of the results [4, 56, 70]. The computational bottleneck is the rendering of shadow images by the rendering engine. The current high-performance graphics hardware on typical desktop computers can render hundreds of images per second, while our optimization algorithm require 40,000 to 80,000 shadow images per trial. Another bottleneck is set operations between shadow images for the evaluation of $E_{coverage}$ in Equation (3.3). There is room for performance improvement since GPU implementation of pixel-by-pixel image operations would achieve significant speedups.

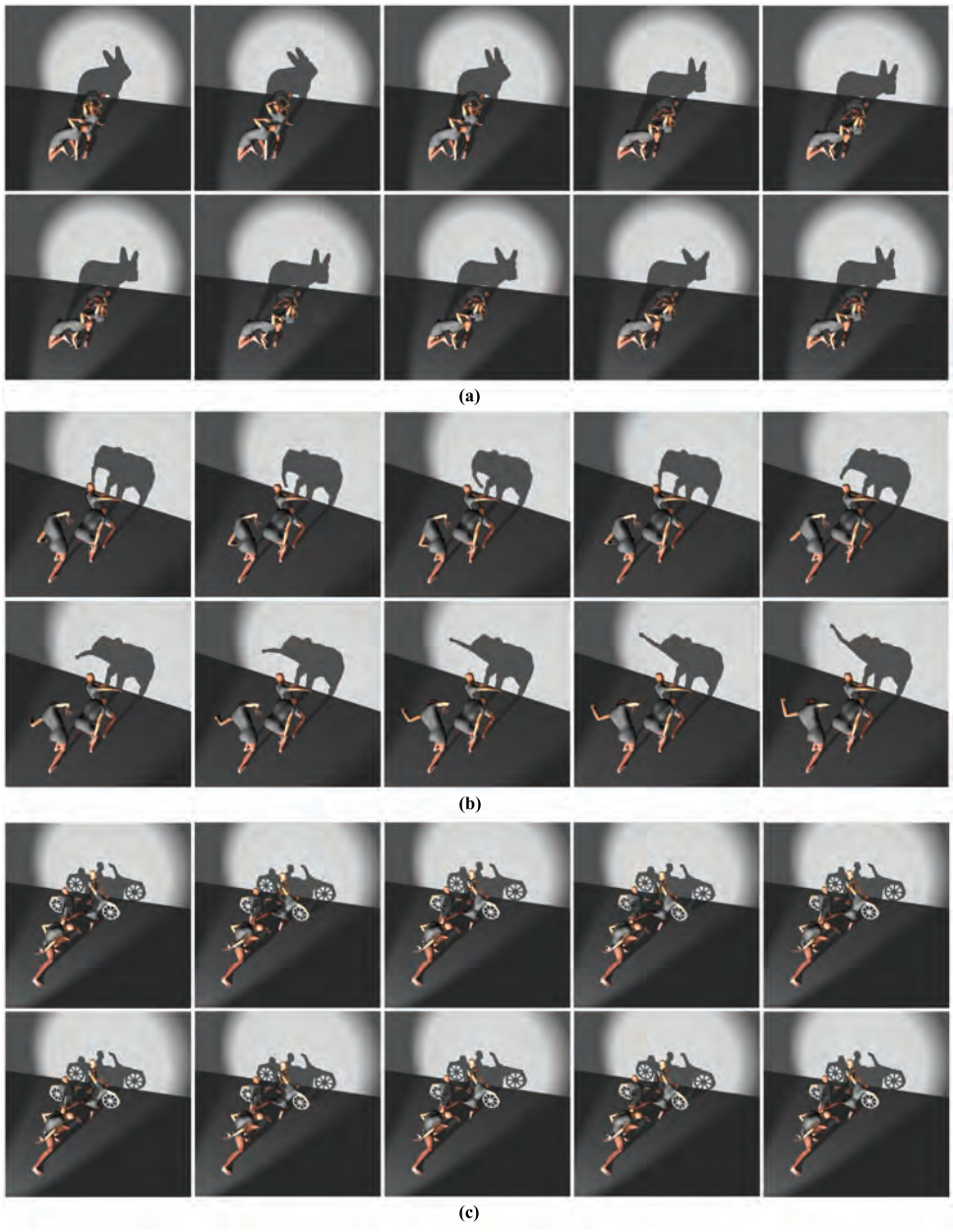


Figure 3.10 Shadow animation. (a) Rabbit. (b) Elephant. (c) Car with a driver.

Chapter 4

Interactive Animation System for Pre-visualization

4.1 Overview

Written or verbal scene descriptions are often quite high-level and sparse. The scriptwriter, director or choreographer specifies only the key events in the scene and omits details such as the exact location, style and timing of each individual action. To aid communication and provide the full choreography, visual aids such as sketches, miniatures, and simple animations are often employed. Scenes with multiple characters are particularly difficult to choreograph as the timing and details of the events are critical to advancing the story in a convincing fashion.

The aim is to facilitate a choreographer's work process by efficiently and intelligently proposing a selection of diverse and high quality scenes that satisfy a desired high-level description, which can be iteratively refined as required by generating new or modified candidate scenes with every iteration. The user can rapidly and semi-automatically choreograph complex scenes with multiple characters interacting from sparse graphical specifications, and iteratively modify these scenes at interactive rates.

To achieve the goals, we present a novel *generate-and-rank* approach that takes

as input a high-level scene specification, in the form of a graph built of sentence-like structures consisting of verbs, subjects, and objects. In practice, this graph could be manually built based on the instructions of a director or animator, automatically extracted from a written text (e.g., a screenplay) or even specified through sketches or gestures. From this specification we automatically generate a large number of plausible scenes from a database of annotated motion capture clips, and then efficiently rank these candidates in order to recommend a small and diverse selection of the highest quality scenes.

A major challenge that we address is how to best rank the plausible scenes in order to select the subset of candidate scenes to present to the user. The most obvious consideration is of course quality, but ranking based on this factor alone will not meet the needs of most users, as the set of highest quality results could all be very similar and therefore not provide enough choice. Therefore, three further closely-related criteria are equally important: the *diversity* of the selection; how much *coverage* is provided of the full space of possible solutions; and how representative each selected scene is of a larger cluster of scenes, i.e., its level of *similarity* to other, non-selected, scenes. Even though animated multi-character scenes are structurally quite different from webpages or images, the ranking approach is conceptually the same, in that prior information about individual items and the relationships between them are utilized to determine the order in which results are presented and thereby reduce redundancy. We therefore adapt some ideas from several modern ranking algorithms for our purposes [8, 41, 73].

In order to generate plausible candidate scenes and evaluate their quality, we evaluate priors and metrics such as how many individual clips are needed; how much they need to be edited in order to satisfy the description; how well the interactions are coordinated; and the extent to which inter-penetrations between characters are avoided. The relative weights of these priors are set by the user based on their quality requirements. For example, the overall staging of the scene is important than motion glitches or inter-penetrations in the early choreography stage, whereas higher motion

quality might be required in the latter stage. Additional quality measures could easily be added, such as saliency or other perceptual metrics.

Our novel generate-and-rank approach for complete motion generation for a scene, borrowing new ideas from other domains such as web and image searching, provides users with a diverse set of the highest quality options from which they can select and refine those that best suit their needs. We support the creativity of the scene choreographer by allowing them a large degree of choice and flexibility in creating their desired scenario, while ensuring that the final result will contain consistent and high quality interactions between the characters. Four technical innovations are presented:a graphical model for high-level scene specification; an efficient and intelligent scene ranking process; a probabilistic motion synthesis algorithm that analyzes the specification to understand the structure of multi-character interactions in individual scenes; and finally, several refinement options that allow the user to have detailed control of the results.

4.2 Graphical Scene Description

The goal of our authoring system is to generate a variety of action scenes based on a high-level written or verbal description (e.g., Figure 4.1a), which should specify the individual motions of the characters, what events occur, and how the characters interact with each other. We provide a graphical user interface to design a diagram of the scene description, called an *event graph* (Figure 4.1c(i)), which exploits subject-verb-object structures to describe events. (While we do not currently perform any natural language processing to automatically create the specification from a written script, this would be a useful enhancement.) In this event graph, actions are represented by *event nodes* if those actions can be described using a clause with a verb, one or more subjects, and possibly objects and phrases. The directional edges of the graph describe the temporal sequencing of events and the transitions of an character from one event to the other. Each event node can have multiple incoming and outgoing edges that correspond to the characters' actions; either both characters are actively

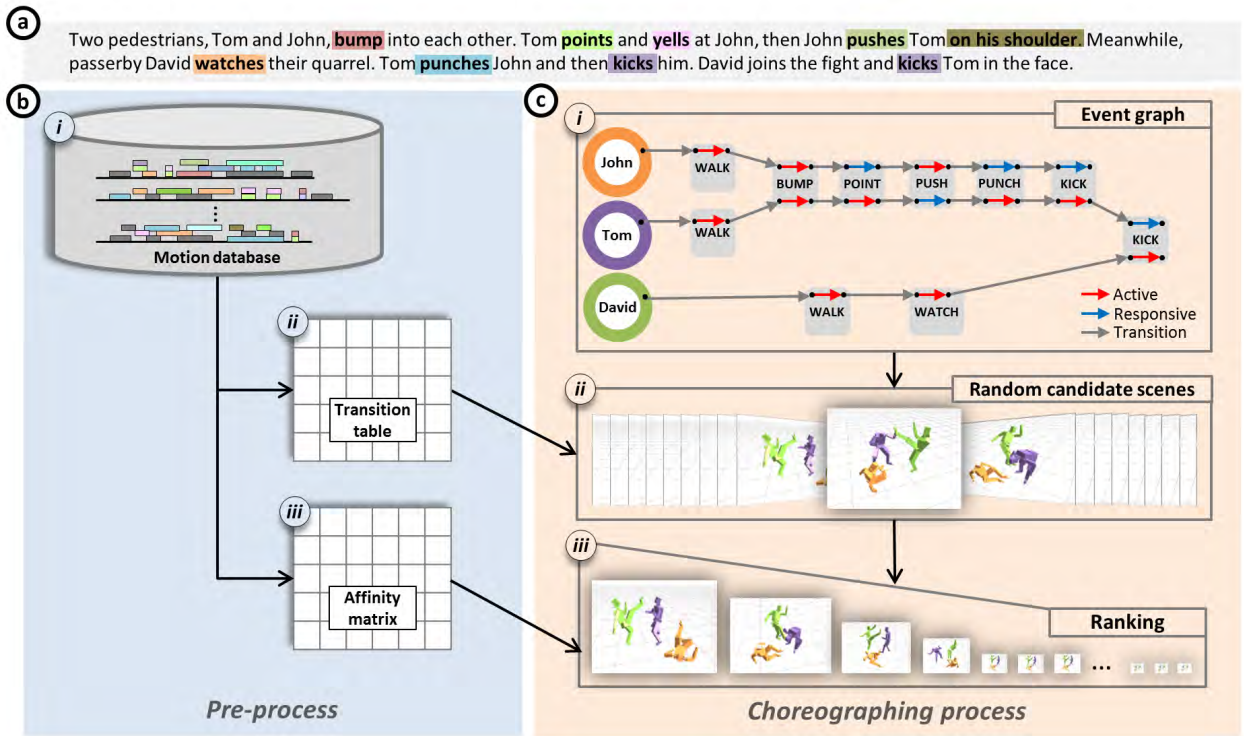


Figure 4.1 Overview of complete scene generation system. (a) The text script of a fight scene. Action verbs and phrases are highlighted. (b) Our motion databases have its frames labeled using our system vocabulary. We also pre-compute a collection of random connecting paths for every pair of action verbs to build a *Transition Table* (ii), and an *Affinity Matrix* (iii) to encode the similarity between all pairs of motion clips. (c) At runtime, the user designs an event graph that matches the text script. Our system generates a collection of random candidate scenes from the event graph and ranks them to present a few top-ranked plausible scenes to the user.

involved, or one is *active* (red arrows in event nodes) and the other *responsive* (blue arrows in event nodes). We currently only allow each event to involve one or two characters for ease of implementation and clarity of presentation, and leave the case of event nodes depicting simultaneous interactions between three or more characters for future work.

To provide high quality, data-driven animation of the events described in the scene, a comprehensive database of suitable motions should be constructed. For the fighting scenarios we use as a demonstration of our approach, we captured one hour's worth of data from professional stunt-men and actors (Figure 4.1b). Motion capture sessions were conducted following standard procedures. We provided twelve high-level themes (e.g., men fighting over a woman, burglar in house, classroom bully, big vs. small guy.) to the actors, which they interpreted and choreographed themselves and also performed some free-style acting/stunt scenarios. Any general or other domain-specific databases could also be used as long as they contain a sufficient variety of relevant interactions. We manually label the motions of each individual motion captured actor using an appropriate vocabulary (though this process could be automated using existing motion classification approaches) consisting of verbs (e.g., bump, push), adverbs (e.g., gently) and phrases (e.g., fall-down). A single motion clip for an actor may have multiple labels (e.g., slap, gently, on-the-face). From viewing the original interaction pairs, the constituent motion clips are labeled appropriately (e.g., slap-active, slap-responsive) as it is possible to pair an active motion with a responsive one that was not simultaneously captured. We also identified physical contacts between body parts (e.g., fist on opponent's face) and the environment (e.g., foot on ground) in a semi-automatic manner and added this information to each motion clip's annotation. Initially, thresholding on distance and relative velocity are used to detect contacts, for which we rectify any false detection manually. More general motion clips (e.g., wandering) are labeled as 'Idle', and are used to make connecting paths between scene events. The plausibility of these 'filler' motions could easily be enhanced by adding new labels to refine the annotations.

From the labeled data we build a *motion graph*, which describes the connectivity between motion clips in the database [54]. Traversing the graph allows longer motion sequences to be generated. In Figure 4.1c(i), the character John follows a sequence of event nodes (Walk, Bump, Point, Push, Punch, Kick). Motion clips with an action verb are called *event clips* and can be associated with an event node that has the same verb label. All plausible motion sequences for John should therefore begin with a Walk event clip, followed by the other event clips in order. Each pair of subsequent event clips may be connected through a sequence of Idle motion clips. From the event clips and their rich connectivity, a large number of plausible motion sequences for each scene character can be generated that satisfy the event graph with respect to label matching. Combinations of these motion sequences for all scene characters constitute a plausible scene if their relative position, direction and timing match when interactions and physical contacts occur. The process of generating the candidate scenes is detailed in Section 4.3. A summary of some definitions and notations we use in this and subsequent sections is provided in Table 4.1.

4.3 Candidate Scene Generation

The successful application of our *generate-and-rank* approach relies on our ability to generate a large collection of plausible candidate scenes. Naïve random walks in the motion graph rarely generate convincing motions because multiple characters will not appear coordinated and too many transitions can result in low-quality motion sequences. In this section, we present a new algorithm to produce a variety of multi-character motion sequences that satisfy the event graph.

Our synthesis algorithm is probabilistic and thus we cannot guarantee that each individual scene generated by our algorithm is always plausible with respect to all conditions and requirements. We present two ideas to alleviate the problem. The first idea is our cycle analysis of the event graph, which allows us to pick plausible scenes with high probability. Secondly, many candidate scenes are generated in the *generate-and-rank* framework and top-ranked scenes among them are highly likely to

Name	**Description**
Character c_i	scene character
Event Node n_i	single- or multi-character event
Event Graph	connects event nodes to make scene
Event Clip: e_i	motion clip associated with event node
Event Path: $p_{i,j}$	sequence of event nodes between n_i and n_j
Cycle: $c(p_i, p_j)$	when two paths share start and end nodes
Transforms: T	move **a**ctive and **r**esponsive characters
T_i^a, T_i^r	move a and r characters *through* event clip e_i
$T_{i,j}^a, T_{i,j}^r$	move a and r *between* event clips e_i and e_j
T_i^{ar}, T_i^{ra}	a and r's relative position and orientation in clip e_i
$t_i^a, t_{i,j}^a, t_i^r, t_{i,j}^r$	associated times
Bodies: B	body coordinate systems and parts
B_i^k	coordinate system of character c_i at frame k
$b_i^{k,m}$	position of c_i's body part m at frame k
t_i^k	associated time

Table 4.1 Definitions and Notation

be plausible within error tolerance.

4.3.1 Connecting Paths

Each event node can be associated with many event clips (Figure 4.2). Pairs of (active, responsive) event clips are associated with each event node involving two characters. Consider event clips e_i and e_j associated with two sequential event nodes. A motion path between e_i and e_j through the 'Idle' part of the motion graph creates a seamless sequence of connected motions. There may be many such motion paths available between any two event clips, with each path requiring different amounts of body translation, rotation, and time.

We pre-compute a collection of connecting paths for each pair of event clips to build a transition table (Figure 4.1b(ii)), the columns and rows of which respectively correspond to sources and destinations of transitions between event clips. Each transition (i, j) contains a variety of paths from e_i to e_j that complete within a certain time limit, or is empty if the shortest path takes too long. In our experiments, the time limit is set to four seconds, and we picked up to 300 paths for each transition (i, j). This collection is expected to cover variations in body translation, rotation and timing while performing the transitional motions from e_i to e_j. We also require each individual path to have as few branching transitions through the motion graph as possible, so we order connecting paths by increasing number of branches and picked paths with fewer branches first. In this way, most paths we generated have only a few branches, thereby minimizing jerky motions.

4.3.2 Motion Cascade

Many event clips and their rich connectivity form a underlying structure that is embedded in an event graph. We call this structure a *motion cascade*. In Figure 4.2, we show a simple event graph (a) and its embedded motion cascade (b) for illustration. Event node n_4 is a single-character node, while the others involve two characters, one of whom is active (in red) and the other responsive (in blue). The edges are solid black

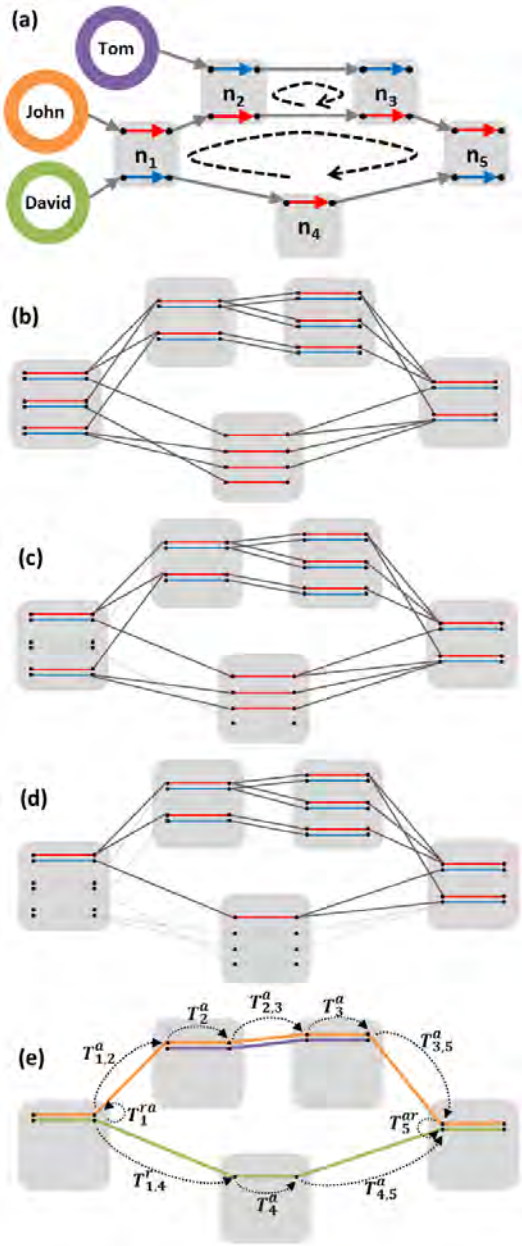


Figure 4.2 Scene Generation: (a) a simple event graph and (b) its motion cascade. We prune event clips that (c) have no outgoing branches and (d) are not cycle-feasible. (e) a candidate scene instantiated from the motion cascade.

if there are any pre-computed paths available between event clips. Even through we draw the edge as a single line, many connecting paths might be available. Given a motion cascade, an instance of a multi-character scene is generated by picking an event clip e_i for each individual node n_i (or a pair of active-responsive event clips for a two-character node) and choosing a connecting path for each individual pair of subsequent event clips.

A sequence of event nodes in the event graph forms an *event path* p_i and, if two such paths share the same start and end event nodes, they form a *cycle* $c(p_i, p_j)$. John (orange) and David (green) traverse paths $p_1 = (n_1, n_2, n_3, n_5)$ and $p_2 = (n_1, n_4, n_5)$, respectively, and their paths coincide at n_1 and n_5 to form a cycle. This means that they interact for event n_1, go their separate ways for other events, and then meet again for event n_5. A pair of active-responsive event clips at the start of a cycle is *cycle-feasible* if they both have connecting paths to the event clips of another pair at the end.

Figure 4.2 also shows that some event clips are 'dead-ends', i.e., either one or both characters have no outgoing branches (c), or they are part of a pair that is not cycle-feasible (d). Such action clips should be pruned before we begin to instantiate our candidate scenes. Whenever any event clip is pruned, we trace its path forwards and backwards to remove any further inconsistencies (e.g., see node n_1 in (c), where the edges of these pruned paths are dotted grey). This process could also introduce a potential dead-end, so building a scene is an iterative process of repeatedly picking event clips and connecting paths one by one from the motion cascade and pruning potential dead-ends.

4.3.3 Motion Selection For Each Cycle

The motion cascade has the potential to produce a very large number of combinations of motion sequences that constitute plausible scenes. However, instantiating one as a candidate is a non-trivial task if the connectivity of the event graph is complex. We first discuss how to generate plausible motion sequences along a single cycle and the

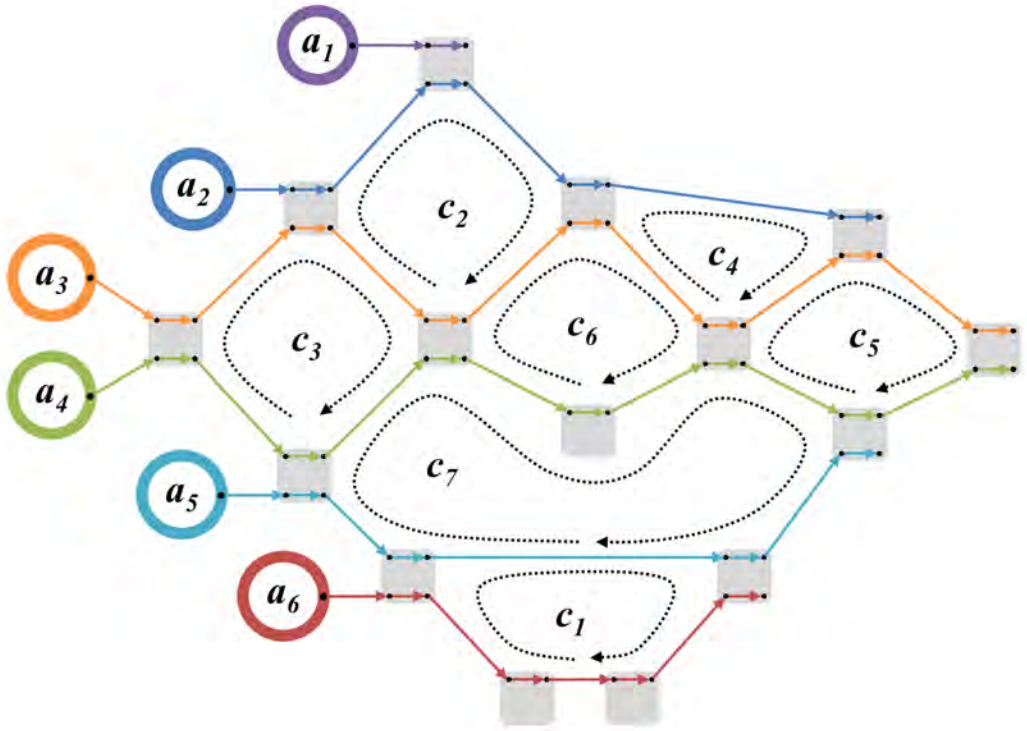


Figure 4.3 Cycle ordering

full version of the algorithm will be presented later in this section.

We define a set of *transforms* for each event clip. T_i^a and T_i^r are 3×3 homogeneous matrices describing two-dimensional translation on the ground plane and rotation about the vertical axis, which bring the body of the active and responsive characters respectively from the beginning of event clip e_i to its end, while $T_{(i,j)}^a$ and $T_{(i,j)}^r$ move the characters *between* event clips. At the start of a pair of active-responsive motions, the relative position and orientation of the interacting characters are represented by the matrix T^{ar}. Similarly, at the end of the active-responsive pair, the relative position and orientation of the interacting characters are represented by the matrix T^{ra}. Times $t_i^a, t_i^r, t_{i,j}^a, t_{i,j}^r$ denote the associated lengths of the aforementioned event clips and connecting paths in time. Therefore, given the cycle for John and David, Equation (4.1) and Equation (4.2) respectively impose spatial coordination and temporal

synchronization between motion sequences for the scene instance in Figure 4.2(e)).

$$T_1^{ra}\ T_{1,2}^{a}\ T_2^{a}\ T_{2,3}^{a}\ T_3^{a}\ T_{3,5}^{a}\ T_5^{ar} = T_{1,4}^{r}\ T_4^{a}\ T_{4,5}^{a} \tag{4.1}$$

$$t_{1,2}^{a} + t_2^{a} + t_{2,3}^{a} + t_3^{a} + t_{3,5}^{a} = t_{1,4}^{r} + t_4^{a} + t_{4,5}^{a} \tag{4.2}$$

Randomly picking event clips and connecting paths among available options rarely meets the cycle conditions in Equation (4.1) and Equation (4.2). We sample many such motion sequences along two adjoining paths and choose the best pair among them. The best pair of motion sequences thus obtained are highly likely to be plausible within error tolerance if sufficiently many samples are picked and examined. In our experiments, we picked 10,000 random samples for each cycle. The spatiotemporal error is measured using

$$\xi = \|\mathbf{v}' - \mathbf{v}\| + w_a|\theta' - \theta| + w_t|\delta' - \delta| \tag{4.3}$$

where $\mathbf{v} \in R^2$ and $\theta \in R$, respectively, are the translation and rotation components of the left side of Equation (4.1), $\delta \in R$ is the left-hand side of Equation (4.2), $\mathbf{v}'$, θ', δ' are from the right-hand side of the respective equations, and w_a, w_t are weights. In our experiments, we determined the weights $w_a = 1.12$ and $w_t = 0.38$ such that the averages of translation, rotation, and time of all available event clips and connecting paths are normalized.

If one of two adjoining paths is much longer than the other (i.e., more event nodes along the path), we may not be able to find any plausible motion sequences that match, because our system limits the lengths of connecting paths. The solution we use is to insert *pseudo event nodes*, where the character idles for a short while in order to fill in the gaps of our sparse description. In this way, we can generate arbitrarily long connections to match adjoining paths, or to create idling behaviors at the beginning and end of the scene if no description is provided for certain characters.

4.3.4 Cycle Ordering

The event graph, in general, has a number of nested, adjacent cycles, which should be visited in an appropriate order. Consider the event graph in Figure 4.3. If we select

the motions of a_2 and a_3 first to form two cycles c_2 and c_4, and the motions of a_4 and a_5 later to form another cycle c_7, then the motions of the third character pair a_3 and a_4 would have already been decided before we can examine whether their motions are well matched around middle cycle c_6. When we visit each individual cycle, at least one side of the cycle should remain undecided so that we retain the freedom to coordinate interactions between characters.

Algorithm 2 shows the overall process for producing a candidate scene, and we now discuss the cycle ordering in Line 2. Our cycle ordering algorithm chooses the last order cycle first and eliminates it from the event graph, then repeat this until only one remains, which will be the first order cycle. In Figure 4.3, it removes cycles from c1 to c7, while our motion synthesis algorithm will visit them backward from c7 to c1 (Line 4 to 7). We choose a cycle at each iteration as follows: we pick any simple event path $p_{i,j}$ between event nodes n_i and n_j. A simple event path is a path that is only composed of two degrees internal nodes. If there exists an alternative path $p'_{i,j}$ between the two event nodes, then p and p' form a candidate cycle and there exist another cycles which share p' with it except when only one cycle remains. Among many candidate cycles, we pick the one with the shortest shared path and eliminate p at each iteration. Whenever the motion synthesis algorithm visits a new cycle, its p' is already decided and p remains undecided. For example, in Figure 4.3, the path traversed by a_6 forms cycle c_1 with a_5's path. Cycle c_1 will be picked first because it has the shortest shared path of length 1. Then we eliminate a_6's path from the graph and repeat the algorithm until we have no more paths to eliminate. This algorithm can guarantee that there will be a *free path*, for which motion sequences are not determined yet, whenever a new cycle is visited.

4.3.5 Generalized Paths and Cycles

The definitions of event paths and cycles generalize further in two respects. First, a single event path can be traversed by multiple characters in relay. Secondly, a path can be traversed backwards in time. This assumption means that the event graph can

Algorithm 2 Scene generation from a motion cascade

// Preprocessing

Build a table of connecting paths

// At runtime

Decide the order of cycles in the event graph

for # of candidate scenes **do**

    for each cycle **do**

        Prune infeasible event clips

        Pick motion sequences for a free path

    end for

    Motion editing and time warping

end for

be undirected, and we do not need two paths to define a cycle anymore. A generalized cycle can be defined as a single path that traverses the event graph and returns to its start node. Even though this generalization may seem extreme, our algorithm readily works with generalized paths and cycles without major modification, thus allowing the algorithm to deal with a wider range of graph configurations that it otherwise could not handle. Figure 4.4 shows such a case, which does not have two adjoining forward paths, but does have a generalized cycle. The transformations along such a cycle are inter-personal when characters take turns, and inverted when backward edges are chosen, which should result in the identity transformation. The generalized cycle in Figure 4.4 poses spatial and temporal conditions as follows:

$$T_1^{ra}\ T_{1,2}^{a}\ T_2^{ar}(T_{3,2}^{a})^{-1}(T_3^{ra})^{-1}T_{3,4}^{r}(T_4^{ar})^{-1}(T_{1,4}^{r})^{-1} = I \tag{4.4}$$

$$t_{1,2}^{a} - t_{3,2}^{a} + t_{3,4}^{r} - t_{1,4}^{a} = 0 \tag{4.5}$$

With these generalized paths and cycles, we can guarantee that a collection of motion sequences constitute a *plausible* scene if the motion sequences are *plausible* along individual cycles. This property allows us to account for the complex connectivity

of the event graph simply by examining individual cycles one-by-one, and therefore makes our algorithm simple, efficient, and easy to implement.

4.3.6 Motion Editing

Even though interactions are carefully coordinated via cycle analysis, there will still be mismatches in the characters' positions and timings. We use motion editing and time warping to rectify these residual artifacts. Consider a scene consisting of individual characters' motions, each of which is a seamless sequence of motion clips. Interpersonal constraints are defined at motion frames in which interactions and physical contacts occur. Let the indices i and j denote characters, k and l frames. Let B_i^k be the body-attached coordinated system and t_i^k is the time of character i at frame k. Each interaction applies three constraints on the body, body parts and time respectively:

$$(B_i^k)^{-1} B_j^l \left((\tilde{B}_i^k)^{-1} \tilde{B}_j^l \right)^{-1} = I \tag{4.6}$$

$$t_i^k - t_j^l = 0 \tag{4.7}$$

$$\tag{4.8}$$

where the tilde indicates the reference values measured from the original motion capture data. The first constraint ensures that the relative position and orientation between characters are preserved. The second equation favors precise synchronization of an action and its response. We employ Laplacian motion editing by Kim et al. [44] to solve this constrained optimization problem, as it makes smooth changes on a collection of motions to meet the constraints, while minimizing motion deformation.

4.4 Scene Ranking

4.4.1 Ranking Criteria

The plausibility or quality of an animated scene can be quite subjective, depending on the target application and/or the viewer's personal goals and preferences. We take inspiration from webpage and image ranking research and apply some of those

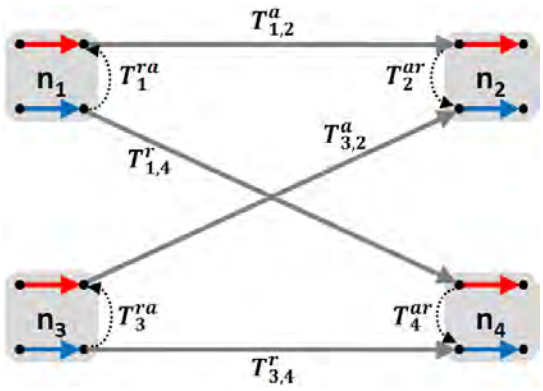


Figure 4.4 A generalized cycle with backwards traversal

concepts to the problem of ranking our choreographed scenes. A common theme that underpins modern ranking algorithms is the characterization of relationships between the items to be ranked. The PageRank algorithm originally exploited in the Google search engine uses hyperlinks between webpages to build a ranking graph [8]. A webpage is considered important if it has many incoming hyperlinks from important webpages. The algorithm determines the ranking of webpages, which correspond to graph nodes, based on prior information (any immediate measure of ranking that might not be fully reliable) and propagation of prior ranking across hyperlinks.

Although images do not have explicit links between them, Jing and Baluja [41] proposed the VisualRank algorithm that creates a link structure among images based on image similarity metrics. An image is therefore considered important if it is similar to important images. We adopt this idea to construct a ranking graph of animated scenes. The similarity between scenes serves as edge weights, and the prior ranking of each individual scene is computed based on our plausibility measure. Propagating prior ranking across visual similarity links results in top-ranked scenes being at the centre of the overall distribution.

Yet another factor we have to consider is the diversity of top-ranked scenes. Unlike images, it is difficult to quickly browse through many candidate scenes, so only a few scenes can be suggested to the user at each iteration of the choreography cycle.

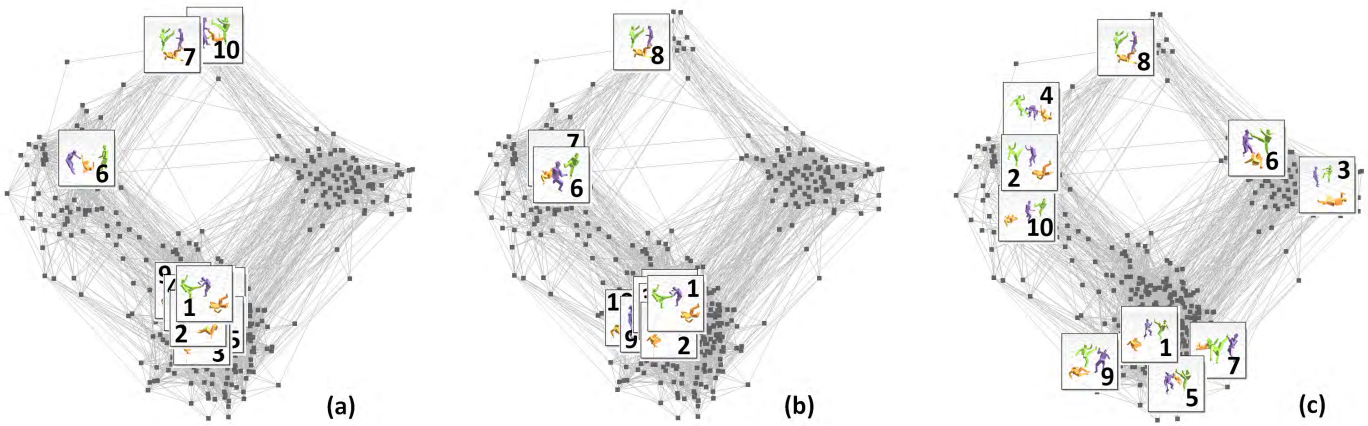


Figure 4.5 Scene ranking. Dimensionality reduction by MDS (multi-dimensional scaling) depicts a collection of 200 scenes in a two-dimensional plane. The Top-10 ranked scenes were chosen based on (a) quality only, (b) PageRank + visual similarity, and (c) our algorithm based on diversity ranking, which demonstrates better diversity and coverage than both others.

Therefore, it is important to have a few top results that are visually different from each other. We employ the idea of diversity ranking [73] that pursues a balance between centrality and diversity in the top ranked results, and also ensures that the full space of candidate scenes is sampled. We can see from Figure 4.5 that the top ten scenes selected using our algorithm based on diversity ranking provides high diversity, as they are less similar to each other; and coverage, as the candidate scenes are sampled more evenly throughout the space of plausible solutions.

4.4.2 Scene Ranking Measures

In order to apply these ideas from modern ranking algorithms, we need some metrics to efficiently evaluate the plausibility and similarity of scenes to facilitate interactive work flow. We therefore define the plausibility P of the scene as the weighted sum of five metrics. Motion editing with multiple actors often involves a trade-off between the degree of deformation and the accuracy of constraint satisfaction. Although allowing large deformations would satisfy all constraints precisely, we often want to limit the degree of deformation to achieve better quality of motion while allowing small residual mismatches in constraints. P_{deform} measures the degree of editing needed to fit motions together for multi-actor coordination, which is calculated using a weighted sum of Laplacian deformation energies for spatial and temporal warping, calculated during motion editing [44]. P_{residual} is the weighted sum of residuals in Equations (4.6)–(4.7) and P_{col} penalizes collisions and inter-penetrations between actors. The diversity metric P_{div} penalizes multiple occurrences of identical actions, because viewers often respond negatively when they spot exactly same actions appearing repeatedly in a single scene. Finally, P_{pref} allows the animator to directly specify his or her preferences on individual actions. In our experiments, the weight values are 2.0 and 0.5 for the spatial and temporal components of both P_{deform} and P_{residual}, 3.0 for collision, 1.0 for diversity, and 0.0 for user preference. In this way, we emphasized the importance of ensuring that characters were in the correct location and collision free, over other factors. However, if a user found diversity to be most important, he could increase

this weight and relax the other constraints.

The animated scene consists of important actions and responses, with more neutral connections between these events. Our similarity measure compares actions and connections at different levels of detail. The similarity between two scenes $\mathcal{S}$, $\mathcal{S}'$ is formulated as follows:

$$\begin{aligned}\text{dist}(\mathcal{S},\mathcal{S}') =& \frac{\sum_i \text{dist}(e_i, e_i')}{(\#\text{ of event nodes})} + w_p \frac{\sum_j \text{dist}(p_j, p_j')}{(\#\text{ of event edges})} + \\ & w_s\left(1 - \frac{|E(\mathcal{S}) \cap E(\mathcal{S}')|}{|E(\mathcal{S}) \cup E(\mathcal{S}')|}\right)\end{aligned} \tag{4.9}$$

$$\text{similarity}(\mathcal{S},\mathcal{S}') = \frac{1}{\text{dist}(\mathcal{S},\mathcal{S}') + \epsilon} \tag{4.10}$$

where $\text{dist}(e_i, e_i')$ and $\text{dist}(p_j, p_j')$ are the dissimilarities between two event clips and two connecting paths respectively. For the computation of action similarity, we use dynamic time warping to align motions in time. The dissimilarity between individual poses is computed based on 39 binary features, suggested by Müller et al. [81]. We compute the similarity between all pairs of event clips to construct an *affinity matrix* during a preprocessing phase (Figure 4.1b(iii)). For connecting paths, a detailed comparison of full-body poses and time alignment is not helpful. Instead, we only compare their spatial transformations T and the length in time using Equation (4.3). The third term indicates the duplication of event clips in two scenes we are comparing. Scene S is composed of a set of event clips $E(\mathcal{S})$ and $|E(\mathcal{S})|$ is its cardinality. If a set of event clips appear in both $E(\mathcal{S})$ and $E(\mathcal{S}')$ in different orders, these two scenes would be perceptually very similar to each other, but the dissimilarity term would not recognize their similarity. The third term compares the duplication of event clips regardless of their ordering.

4.5 Scene Refinement

The ability to refine the top ranked results is an essential component of our generate-and-rank approach, which allows the scene choreographer to have immediate and

detailed control over the results. The user is provided with a range of refinement options through an appropriate user interface. The computation cost for the refinement varies depending on how deep a dive into the hierarchy of the process flow is needed to execute it.

Interactive Manipulation: The user may often find highly ranked scenes to be satisfactory except for small glitches that can be easily fixed. He may want to remove collisions between characters, require an character to face a particular direction or to reach a particular location at a particular time. Any scene generated from our system is readily annotated with interaction constraints. Therefore, we can use Laplacian motion editing while maintaining the integrity of multi-character coordination, which is the most immediate and direct type of refinement we facilitate.

Re-ranking and Re-generation: The user can adjust weights of rank metrics, which are then reflected in the next round of ranking. Re-ranking candidate scenes can be done quickly in about one second. A more expensive option is re-generation, which involves either a change to the event graph, or adding a new set of motion data and labels. In the former case the motion cascade should be rebuilt to generate a new set of candidate scenes, which takes just a few minutes for simpler examples and may take up to an hour for the largest example we tested. The latter, more extreme, change requires the motion database to be updated from scratch, which can take a few hours.

Action Replacement: Given any scene, we might wish to replace a particular event clip (or a pair of active-responsive clips) with another, while leaving the remaining scene intact. The new event clips at an event node will be connected to the remaining part of the scene by choosing appropriate connecting paths along incident event edges. A brute-force approach examines all possible combinations, which takes $O(N^p)$ computation time, where N is the number of available connecting paths and p is the number of (both incoming and outgoing) event edges. If the event node has two characters, action replacement takes $O(N^4)$ time. We suggest a more efficient algorithm of $O(pN^3)$ time complexity. For simplicity of algorithm description, we assume

that a pair of active-responsive event clips in a two-character event node is replaced with another pair (a, r), so $p = 4$. From earlier, T^a and T^r are the associated transformations, while T^{ar} and T^{ra} are the inter-personal transformations between the partners at the start and end of the event clips (Figure 4.6). Characters a and r are supposed to be connected to the remaining part of the scene through four connecting paths. Homogeneous matrix C_i for $1 \leq i \leq 4$ denotes the position and orientation at the end of a connecting path, where it should be incident with either a or r. We need to choose four connecting paths such that the error E_{connect}, i.e., the sum of misalignment distances, is minimized:

$$\begin{aligned} E_{\text{connect}} = \; & \text{dist}(C_1^{-1}C_2, T^{ar}) + \text{dist}(C_2^{-1}C_3, T^r) + \\ & \text{dist}(C_3^{-1}C_4, T^{ra}) + \text{dist}(C_4^{-1}C_1, (T^a)^{-1}) \end{aligned} \tag{4.11}$$

Our algorithm is based on dynamic programming. Assuming that we first choose the n-th connecting path for C_1, deciding the other three paths requires the construction of a table of $p \times N$ fitness values, by solving the recurrence equations:

$$V(1, j) = \text{cost}(1, n, j) \tag{4.12}$$

$$V(i, j) = \min_k V(i-1, k) + \text{cost}(i, k, j) \tag{4.13}$$

where $\text{cost}(i, k, j)$ is the misalignment of choosing the k-th connecting path for C_i and j-th connecting path for $C_{(i+1) \bmod 4}$, assuming that the preceding connecting paths have been chosen optimally. $V(i, j)$ is the accumulated misalignment of choosing j-th connecting path for $(i + 1)$-th event edge assuming that previous choices of connecting paths are optimal. Because of the cyclic ordering of the event edges, index i is modulo 4. The table entries are filled based on dynamic programming. Backwards tracing from $V(4, n)$ identifies a cyclic path through the table. We repeat the dynamic programming for each n to examine all possible combinations and choose the best one that minimizes the error E_{connect} in Equation (4.11). Even though we only explained about spatial coordination of connecting paths, temporal synchronization is

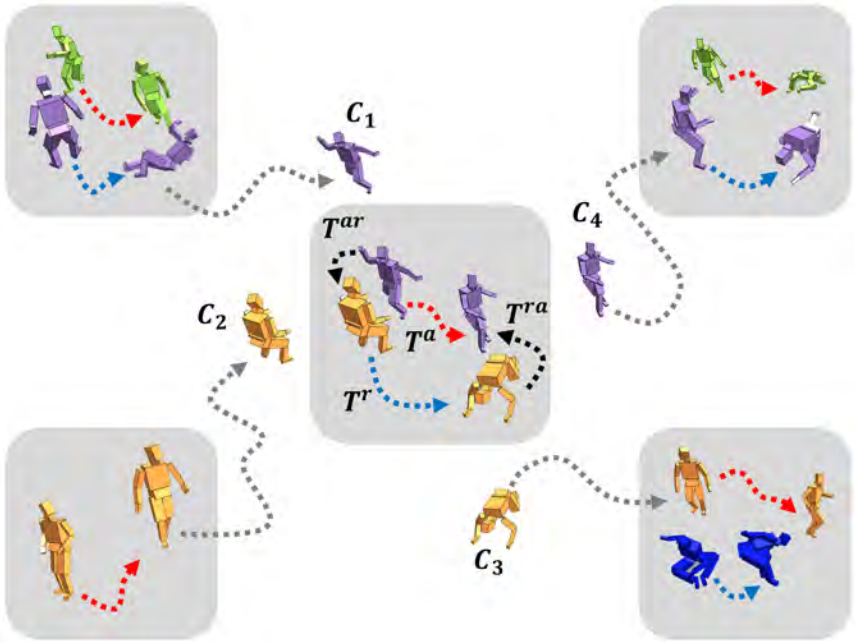


Figure 4.6 Action replacement

also considered to compute E_{connect}. So, the *dist* function of Equation (4.11) is computed by Equation (4.3) using spatial and temporal error. In practice, the brute-force $O(N^4)$ algorithm takes about 10 seconds to find the optimal set of connecting paths, while our algorithm takes less than one second to achieve an order of magnitude performance gain.

Partner Changing: Even though the motion cascade readily provides very many candidate scenes, sometimes richer variability is required for a specific event node, for which a limited number of event clips are available. Partner Changing is the process of combining active and responsive motion clips captured separately in order to enrich variability. Consider two pairs of action-response clips (a, r) and (a', r'). Each pair has inter-personal transformations and corresponding body parts at some frames in which interactions or physical contacts occur. Partner changing generates new crossing pairs (a, r') and (a', r), if there exists a correspondence between interactions in the two original pairs, while corresponding transformations and distance between

body parts are similar within user-specified thresholds. The averages of corresponding transformations serve as interaction constraints for the new crossing pairs. Laplacian motion editing with halfway constraints solves for motion deformations that match crossing pairs well.

4.6 Experimental Results

We demonstrate the power and scalability of our generate-and-rank approach through a variety of examples. Figure 4.7 shows the event graphs for additional examples. Each example has a story of events.

- **Payback:** Three guys are sitting on the ground, stretching and exercising. Another guy bugs and irritates them repeatedly. They all stand up and pay him back for the irritation.
- **Tournament:** Eight people fight in an elimination tournament. The winner goes through to the next round, while the loser falls down and stays on the ground. The final winner cheers for victory.
- **Movie:** We recreate a scene based on a fight sequence from an actual movie (Snatch, ©Columbia Pictures, 2000), where two guys point, yell, punch, kick, grab, and throw each other. Although all our motions had been captured with no reference to this scene, our system was able to emulate the original sequence very well.
- **Random fight:** Twenty actors fight randomly with each other (Figure 4.8). This example demonstrates the scalability of our approach: the event graph includes 268 events and 320 edges, and our algorithm identified 52 cycles in the graph.

Our system generates 1000 candidate scenes for each example. Each candidate scene is between 30 to 80 seconds long and has a very high-dimensional representation (scene-time $\times$ frames per second $\times$ number of actors $\times$ degrees of freedom per pose).

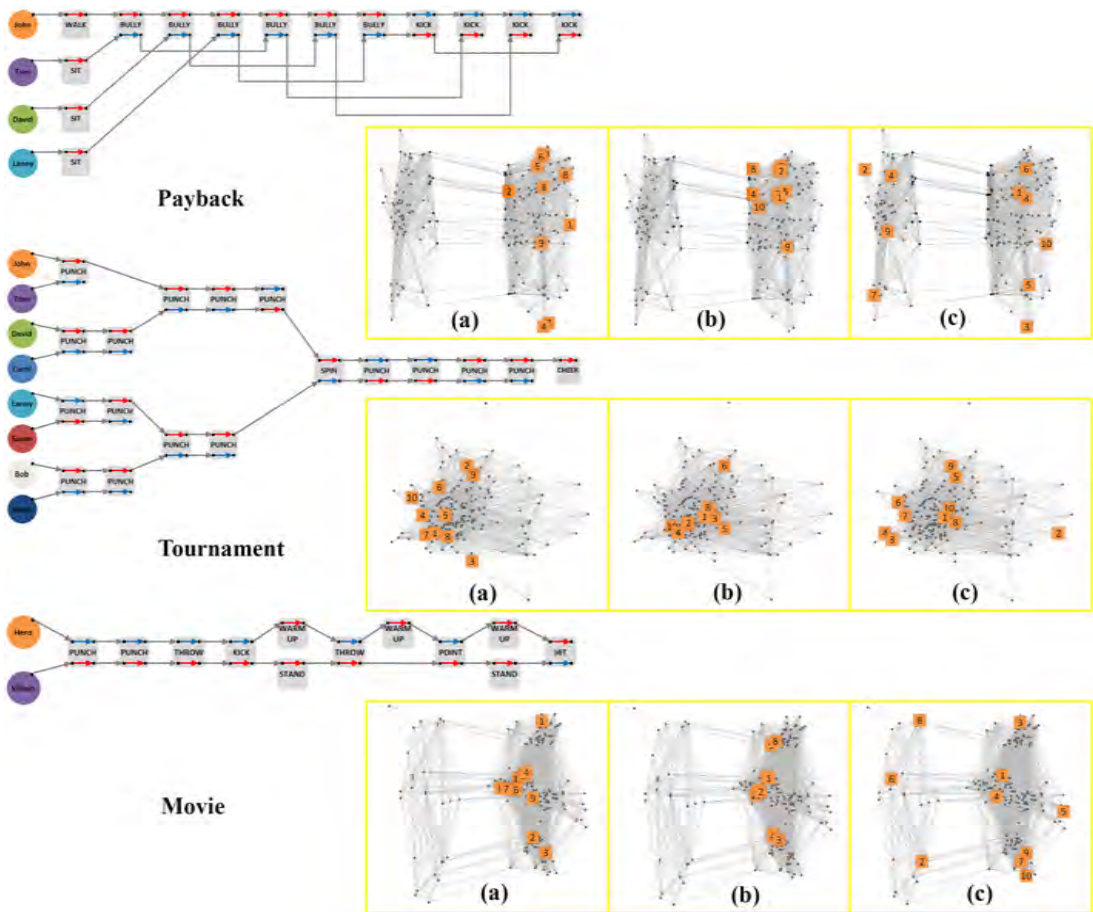


Figure 4.7 The event graph and MDS visualizations of examples. Top ten results are selected by (a) quality only, (b) PageRank + visual similarity, and (c) our algorithm.

Multidimensional scaling (MDS) allows us to visualize the level of similarity in our high-dimensional data. As Figure 4.7 shows, each scene has a clustered distribution. The top ten results selected using quality only and PageRank with visual similarity links tend to be quite similar and do not offer much coverage of all clusters. Our algorithm based on diversity ranking, however, always selects a highly relevant yet diverse set of scenes for the top ranked results, which broadly cover the space of candidate scenes.

Performance statistics are measured on a desktop PC equipped with an Intel Xeon CPU E5-2680 (8 cores, 2.7 GHz) and 32GB main memory, except for the random fight example. Creating such a large scene is memory intensive, so we computed the random fight example on another machine with four processors of an Intel Xeon CPU E7-4870 (2.4 GHz) and 1TB memory. Our database consists of 78,502 frames of motion data cleaned up and labeled. The motion graph constructed from the database includes 70,861 frames (about 40 minutes long) of deadend-free, strongly-connected components. The motion database has 25 verb labels and 20 phrase labels (Table 4.2). In the preprocessing phase, the construction of a motion graph, an affinity matrix, and a table of connecting paths took 5.7, 10.6, and 88.5 minutes, respectively. Therefore, rebuilding these structures from scratch takes about 105 minutes of computation in total.

The runtime computation for each example is summarized in Table 4.3. The units are in meters for distance, radians for angle, and seconds for time, unless otherwise specified. The breakdown of the runtime computation shows that motion editing and motion selection around each individual cycle are the most time-consuming components. Kim et al [44] suggested an acceleration technique based on frame sub-sampling, which we have not yet incorporated into our system, but which we expect will deliver an order of magnitude improvement in performance. The computation time for motion selection depends on the number of samples we pick for each individual cycle. In principle, we have to test more samples for longer cycles since they may provide more diversity of motion choices. Table 4.4 shows how the number of samples may affect

Action verbs			Phrases	
Single Action	***Hit***	***Interaction***	***Body part***	***State***
bend	kick	block	head	standing
cheer	punch	bump	chest	sitting
faint	slap	chase	stomach	bending
lie	***Bully***	face	pelvis	lying
sit	dig	grab	upper leg	***Strength***
stand	nudge	handshake	lower leg	low
walk	press	point	foot	medium
warm up		spin	shoulder	high
		step on	upper arm	***Direction***
		throw	lower arm	left
		push	hand	right

Table 4.2 The vocabulary of action labels

the motion quality for the three-person example. The quality improves as the number of samples increases and the improvement plateaus at about 10,000 to 20,000 samples per cycle. Currently, we pick 10,000 samples per cycle regardless of its length, and there are opportunities for further improvement by picking samples adaptively.

4.7 Discussion

Even though choreographing fight scenes are on presented as examples, our approach could be easily extended to deal with other types of scenes where there is a requirement for the coordination of multiple interacting actors, such as dancing, sports or social interactions. We focused on fighting scenes because they are particularly difficult to synthesize and therefore present a significant challenge. The problem becomes simpler the fewer physical contacts there are between the actors. Instead, other factors, such as facial expression, lip synch, gaze direction and gesture, would become more important. These factors have been the subject of many studies and are highly complementary to and compatible with our approach.

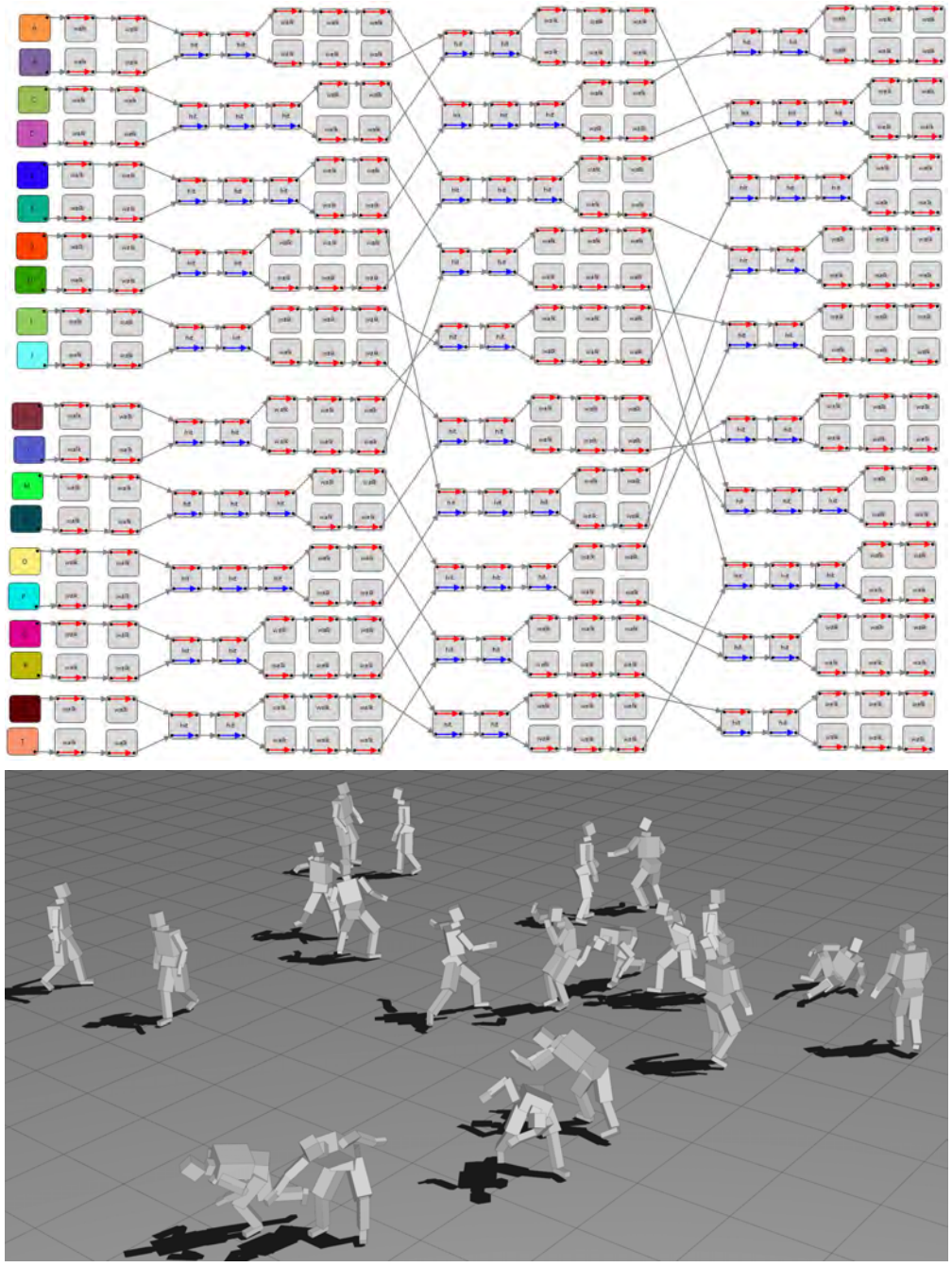


Figure 4.8 Random fighting

		Three people	Tournament	Payback	Movie
Graph size	Persons	3	8	4	2
	Event nodes	10	18	14	12
	Event edges	13	27	19	17
	Cycles	4	11	7	6
	Pseudo nodes	2	16	39	18
Play time		33.2	59.8	57.1	83.4
Time	Cycle ordering	0.014	0.061	0.057	0.022
	Pruning	0.255	21.201	5.426	0.200
	Motion selection	252.907	1183.268	1913.907	718.634
	Motion editing	78.376	1055.058	2624.184	529.917
	Ranking graph construction	2.663	7.863	9.306	5.503
	Ranking	0.361	0.422	0.357	0.350
	Total	334.576	2267.873	4553.236	1254.626

Table 4.3 Runtime performance

Our system currently can handle up to two actors for each event. This limitation is not inherent to our approach, but was rather chosen for convenience of system implementation and clarity of exposition. In principle, our algorithms can readily handle more general forms of events with arbitrarily many subjects and objects. Similarly we only handled the interaction which is instantaneous, but it could be extended easily to handle continuous interaction by adding continuous constraints where interactions occur to the laplacian motion editiong formulation. We need additional formulation for partner changing accordingly.

Another promising direction for future work is to incorporate physics simulation into our generate-and-rank framework. Physics simulation can provide rich variability and realism without expanding the size of motion databases. As shown by Twigg and James [108], generate-and-browse is a viable option for steering complex multi-body dynamics simulations, from which our system could benefit greatly in order to generate more realistic interactions and collision effects. The perceptual plausibility of the causality between the selected action-response motion pairs is also an important factor that should be evaluated and taken into account, as shown by Hoyet and

# of connecting paths	Total scenes		Top ranked scenes	
	Spatial	Temporal	Spatial	Temporal
100	0.04460	0.00019	0.01206	0.00011
300	0.03002	0.00020	0.00592	0.00005
500	0.02995	0.00018	0.00515	0.00005
700	0.02740	0.00024	0.00321	0.00003
900	0.03071	0.00023	0.00243	0.00006

Sampling per cycle	Total scenes		Top ranked scenes	
	Spatial	Temporal	Spatial	Temporal
1	0.69220	0.01923	0.38662	0.00499
100	0.22817	0.00295	0.05972	0.00075
1,000	0.11445	0.00092	0.02875	0.00032
10,000	0.06182	0.00038	0.00802	0.00008
100,000	0.03002	0.00020	0.00592	0.00005

Table 4.4 Motion quality with respect to the number of connecting paths and cycle sampling for the three-person example. Spatial and temporal Laplacian energies normalized per frame indicate the degrees of spatial deformation and time warping. Low energies indicate better quality results. The average energies over all 1000 candidate scenes and the top 8 scenes are provided.

colleagues [36], and physical simulation could also enhance perceived visual quality.

Sound effects are also important elements of scene choreography. For the examples described in this paper, we manually specified sound effects. A straightforward extension would be to include audio information in the motion database, allowing us to synthesize sound effects to match the constructed scenes. Incorporating domain expertise into the design of our motion databases would also be very valuable, in the same way that Calvert and Ma [10] incorporated the input of expert dance choreographers.

Chapter 5

Physics-based Design and Control

5.1 Overview

The beauty of flapping flight comes from the complex interaction of gravity, muscle actuation, and aerodynamics. We expect that winged creatures are able to perform a variety of motor skills including soaring, gliding, hovering, taking off, and landing, though developing such a motor skill is still a daunting task. Even with an imaginary creature, its motor skills are desired to be physically realistic and controllable in a way that its wingbeats generate lift/thrust force and steer the flying direction. Balancing during flapping flight is far more challenging than fixed-wing flight because the dynamic model of the creature is usually under-actuated. This means that the dynamic system has more degrees of freedom than it can actuate. The control of an under-actuated system is inherently ill-posed.

At any state s of the creature and its action a, the dynamic system brings the creature to a subsequent state s'. Construction of a motor skill (controller) can be viewed as building an inverse mapping function that answers action a given current state s and desired state s'. During runtime the desired state can be determined directly or indirectly according to environments and applications. The motor skill should achieve

three criteria. The first criterion is maneuverability, which means that it can fulfill user-provided goals (for example, reaching to a target location or following a path). The second criterion is balance. Generated actions should make the creature maintain its balance. The final criterion is naturalness of motions. For example, motions of the creature should not be jerky. Ju et al [42] proposed a data-driven approach that utilizes real motion capture data of birds as a base data, then generates random samples around the data to construct a stable controller. Although the three criteria were achieved successfully, it can not be applied to imaginary creatures whose motion capture data are not available. As extended work of [42], we present two methods that can apply to imaginary creatures. The first method generates state-action pair data by running a collection of trajectory optimizations for a cycle of wingbeat, and then constructs KNN (K Nearest Neighbor) regression model from the data, whereas the second method constructs Deep Neural Networks (DNN) and learns using Deep Q-Learning (DQL).

5.2 Combining Regression with Trajectory Optimization

Recent progress in nonlinear, non-convex optimization methods, however, facilitates the construction of motor skills of an under-actuated system. Given a specific target (for example, a target location to reach in a single cycle of wingbeat), a motor skill generates torques at actuated joints in an open-loop simulation to bring the dynamic system to the target location. A motor skill may have control parameters, which span a range of targets. Given any target in the range, a dynamic system with a parameterized motor skill can generate an appropriate control strategy responsively at runtime.

The parameterized motor skill (or controller) has an underlying representation of its dynamics and control strategies. The representation can be either parametric (for example, linear dynamic models) or non-parametric (for example, random samples around a nominal trajectory). In either case, the construction of a parameterized skill requires a large collection of open-loop simulation trials with a random/regular grid

of targets. Fitting a dynamic model to the trials or regression over the trials allows for parameterized control of animal locomotion. The non-convex optimization for an individual open-loop simulation is computationally expensive and thus the controller construction is even far more demanding.

We present a new method for constructing parameterized motor skills for animal locomotion of arbitrary morphologies and simulation environments. In particular, we created three synthetic creatures: a luxo hopping on the ground, a turtle swimming under the water, and a bird flying in the air. We begin with a user-provided, hand-crafted, base controller that simply repeats a reference trajectory without any modulation. The reference trajectory may be procedurally-defined, keyframed, or motion captured. Our algorithm automatically generates a parameterized motor skill from the based controller, providing improved maneuverability and interactive performance. We demonstrate that our creatures can be physically simulated in realtime and interactively controlled while being responsive to external perturbation.

Our construction algorithm also requires a number of simulation trials to optimize the motor skill. The key idea of efficient computation is to exploit the coherence of optimization problems we need to solve. Solving a system of coherent optimization problems can be more efficient than solving individual optimization problems independently. Our recollective CMA algorithm incrementally accumulates the traces of optimization processes, which serve as prior knowledge for solving subsequent optimization problems more efficiently. Our experimental results show that the use of cache samples not only achieves substantial performance gain, but improves the convergence of optimization.

5.2.1 Simulation and Motor Skills

The dynamic system has n_a actuated degrees of freedom and n_p unactuated degrees. The state S is a $(n_a + n_p)$-dimensional vector. We want to simulate its locomotion. The motor skill of the system is a specific set of torque profiles at actuated joints for a cycle of locomotion. The open-loop simulation brings the system from its initial

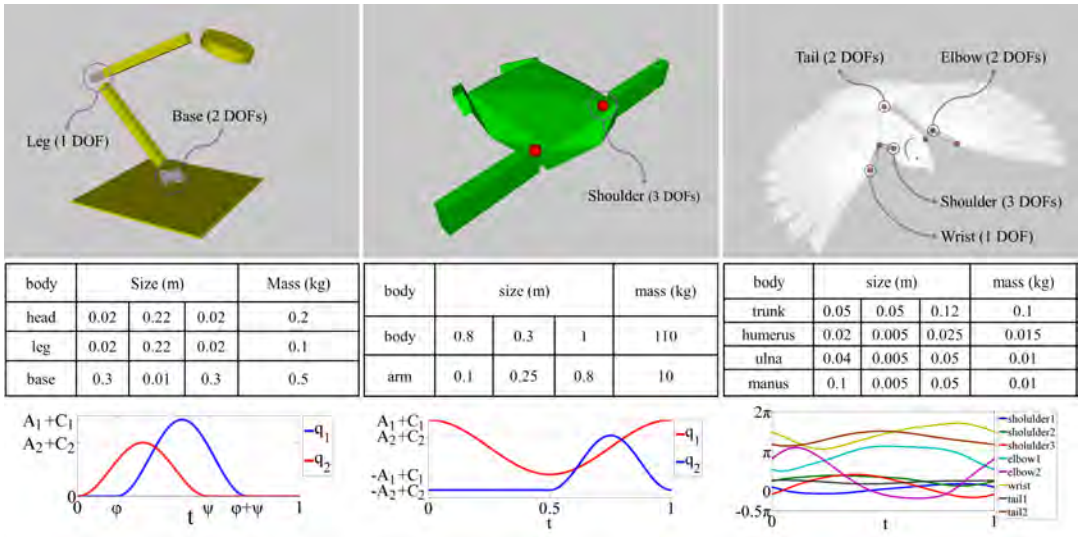
Figure 5.1 Physically based simulation and interactive control of synthetic creatures: (left to right) a luxo hopping on the ground, a turtle swimming underwater, a dove and a peacock flying in the air

state S to a new state S'.

The motor skill is a dynamic controller that can be defined in many different ways. It may have a shaping function that describes how actuated joints move. The shape function is either specified manually by using sinusoidal functions or provided as a reference trajectory, which can either be motion-captured or keyframed. Alternatively, the torque profiles can be directly specified as step functions. It is not important how the motor skill is defined as long as we have a small number of skill parameters (for example, the coefficients of sinusoidal functions and the control points of reference trajectories) to maneuver. The motor skill works only on actuated joints and does not generate any feedback/feedforward signals to modulate unactuated joints.

We tested our algorithm with three synthetic creatures: A luxo jumping on the ground, a turtle swimming under water, and a bird flying in the air. Their motor skills are designed to have as few skill parameters as possible while allowing them enough maneuverability. The simplest creature, the luxo, has three controllable parameters, while the most complex, the bird, has 29 parameters. The task space of animal locomotion spans a range of location and steering angles that can be reached in a single cycle of locomotion. The tasks are parameterized in an egocentric coordinate system. The subscripts L, T, and B indicate parameters for our Luxo, Turtle, and Bird, respectively.

Jumping Luxo. The luxo is an articulated dynamic system with three rigid bodies (head, leg, base) connected by a hinge (head to leg) and universal (leg to



body	Size (m)			Mass (kg)
head	0.02	0.22	0.02	0.2
leg	0.02	0.22	0.02	0.1
base	0.3	0.01	0.3	0.5

body	size (m)			mass (kg)
body	0.8	0.3	1	110
arm	0.1	0.25	0.8	10

body	size (m)			mass (kg)
trunk	0.05	0.05	0.12	0.1
humerus	0.02	0.005	0.025	0.015
ulna	0.04	0.005	0.05	0.01
manus	0.1	0.005	0.05	0.01

Figure 5.2 Simulation models: (Left) Luxo, (Middle) Turtle, and (Right) Bird. The middle row shows the size and mass of individual body parts. Their inertia matrices are estimated from their bounding boxes. The bottom row shows the normalized reference trajectories for their base controllers. The reference trajectories for the luxo and the turtle are generated using sinusoidal functions, while our bird exploits motion capture data as its reference trajectories.

base) joints. The full state S_L of the luxo has nine DOFs that include the position and orientation of the base and three DOFs at joints. We parameterize the reference trajectory of actuated joints as periodic cycles in generalized coordinates. The luxo tracks the reference trajectory by using PD servos. Let q_1 be the joint angle of the leg and let q_2 and q_3 be the joint angles at the base.

$$q_1 = \begin{cases} A_1 \sin\left(\frac{2\pi}{\psi}(t-\phi) - \frac{\pi}{2}\right) + C_1 & \text{if } \phi \le t \le \phi + \psi, \\ -A_1 + C_1 & \text{otherwise} \end{cases} \tag{5.1}$$

$$q_2 = \begin{cases} A_2 \sin\left(\frac{2\pi}{\psi}t - \frac{\pi}{2}\right) + C_2 & \text{if } 0 \le t \le \psi, \\ -A_2 + C_2 & \text{otherwise} \end{cases} \tag{5.2}$$

We use an identical form of equations for q_2 and q_3. t is normalized time varying from zero to one. Both functions are bell-shaped, but their phases differ by ϕ (see

Figure 5.2 bottom left). q_1 spans interval $[\phi, \phi + \psi]$ and q_2 spans interval $[0, \psi]$. A_i and C_i are their amplitudes and offsets, respectively. ϕ, ψ, and C_i are constant in our experiments. The luxo has four skill parameters $X_L = (A_1, A_2, A_3, T)$, where T is a timewarp factor. The duration of a locomotion cycle is scaled by T, which makes the gait faster or slower than the reference. The luxo is supposed to be able to hop in any direction on the horizontal plane. The task parameters $Y_L = (x, z)$ represent a displacement vector it travels for a single hopping.

Swimming Turtle. The turtle has a body and two arms, which are connected to the body through three-DOFs, ball-and-socket joints. The full state S_T of the turtle has 12 DOFs that include the position and orientation of the body and six DOFs at arm joints.

$$q_1 = A_1 \sin\left(2\pi t + \frac{\pi}{2}\right) + C_1 \tag{5.3}$$

$$q_2 = \begin{cases} A_2 \sin\left(4\pi t - \frac{\pi}{2}\right) + C_2 & \text{if } \ 0.5 \le t \le 1 \\ C_2 & \text{if } \ 0 \le t \le 0.5 \end{cases} \tag{5.4}$$

q_1 corresponds to the dihedral angles of the arms. The reference trajectory of the sweep and twist angles use Equation (5.4). The joint angles vary periodically between its lower bound $-A_i + C_i$ and upper bound $A_i + C_i$ (see Figure 5.2 bottom middle). The skill parameters include A_i and C_i for each DOF and a timewarp factor T. Therefore, the turtle has 13 skill parameters, $X_T = (A_1, C_1, \cdots, A_6, C_6, T)$. Its task parameters $Y_T = (x, y, z, w_x, w_y, w_z)$ include linear and angular displacements with respect to the body-attached, moving coordinate system. This means that a single cycle of flapping makes the turtle move forward by (x, y, z) while steering its direction. The total steering for the cycle is represented as three-dimensional rotation about the axis of (w_x, w_y, w_z) by the angle of its length.

Flying Bird. The aerodynamics of bird flight entails complex interactions among a bird's muscles, skeleton, and feathers. We use a dove model presented in [42]. The bird model has an articulated skeleton of rigid bones and flexible feathers. The skeleton consists of its trunk, humerus (upper arms), ulna (lower arms), and manus

(hands). The bones are connected by joints at the shoulders, elbows, and wrists. The shoulder is a ball-and-socket joint with three DOFs, the elbow is a universal joint with two DOFs, and the wrist is a hinge joint with one DOF. The dove has ten primary and eight secondary feathers on each wing, and twelve tail feathers. The primaries are connected to the manus, while the secondaries are connected to the ulna. The feathers play an important role in flapping flight, as they are the principal source of lift and thrust, supporting the bird and moving it forward through the air. The tail feathers are actuated elements that can tilt up/down and fan in/out actively, helping the bird to brake and steer in flight. Each individual feather is a triangular mesh that can bend and twist flexibly. The dove model has 14 actuated DOFs in total (6 DOFs for each wing and 2 DOFs for tail feathers). We can define the reference trajectories using sinusoidal functions, similarly to the turtle. Alternatively, we use motion capture data acquired from a real dove in flight to demonstrate the flexibility and versatility of our approach (see Figure 5.2 bottom right). The motion capture data were downloaded from a public motion database [101]. From a computation point of view, it does not matter if we use sinusoidal functions or motion capture references because both would end up with an abstraction with skill parameters. We parameterize the motion capture reference trajectory for each DOF by using upper and lower bounds, U_i and D_i. The reference trajectory transforms linearly according to the bounds. The bird has 29 skill parameters, $X_B = (A_1, C_1, \cdots, A_{14}, C_{14}, T)$. Its task includes six parameters $Y_B = (x, y, z, w_x, w_y, w_z)$, where (x, y, z) and (w_x, w_y, w_z) are linear and angular displacements, respectively.

5.2.2 Control Adaptation

A tuple of skill parameters $X = (x_1, \cdots, x_N)$ determines what task Y the controller performs. Given any controller, control adaptation is a process of adjusting the skill parameters to perform a new task Y'. We formulate control adaptation as an energy

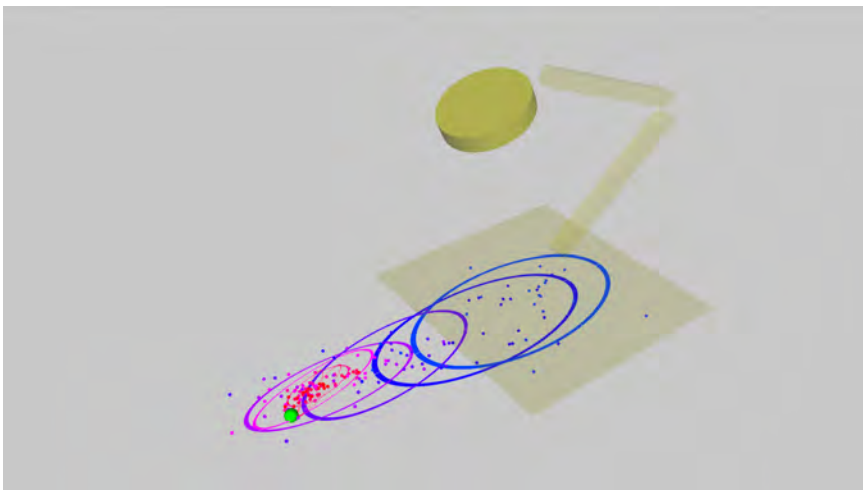


Figure 5.3 Control adaptation via CMA optimization. The base controller makes the luxo hop in place. The optimized controller makes it jump forward to the target location, shown as the green ball. The dots and ellipses on the ground show the traces of samples in CMA. Each dot represents the landing location of a sample of jump simulation. Each ellipse shows the covariance of landing locations at each generation. The color changes from blue to red as the iteration proceeds.

minimization problem. The fitness function includes three terms.

$$E = E_{\text{task}} + E_{\text{effort}} + E_{\text{regul}}. \tag{5.5}$$

The first term E_{task} favors a new set of parameters that better performs the desired task.

$$\begin{aligned} E_{\text{task}} =& c_p \|\mathbf{p}_d - \mathbf{p}\|^2 + c_q \|\log(\mathbf{q}^{-1}\mathbf{q}_d)\|^2 + \\ & c_v \|\mathbf{v}_d - \mathbf{v}\|^2 + c_w \|\mathbf{w}_d - \mathbf{w}\|^2, \end{aligned} \tag{5.6}$$

where $\mathbf{p}$, $\mathbf{q}$, $\mathbf{v}$, and $\mathbf{w}$ are the position, orientation, linear velocity, and angular velocity, respectively, of the dynamic system. The subscript d indicates the desired value for performing the task. $\|\log(\mathbf{q}^{-1}\mathbf{q}_d)\|$ is the geodesic distance between unit quaternions $\mathbf{q}$ and $\mathbf{q}_d$ [51]. The second term minimizes the effort for performing the task.

$$E_{\text{effort}} = \sum_k \int_0^T w_k \|\tau_k(t)\|^2 dt, \tag{5.7}$$

where τ_k is the torque exerted at joint k. w_k's are weight values for individual DOFs. E_{regul} is a regularization term that prevents the optimization from drifting away from initial parameters.

$$E_{\text{regul}} = c_{\text{regul}} \|X_0 - X\|^2, \tag{5.8}$$

where X_0 is the initial parameter. The evaluation of the fitness function for a new set of skill parameters requires an open-loop simulation of the dynamic system.

Algorithm 3 Standard CMA algorithm

N_g : maxinum # of generations

N_s : # of samples in each generation

set μ_0 and Σ_0
for $i \leftarrow 1, N_g$ **do**
    for $j \leftarrow 1, N_s$ **do**
        $X_i^j \leftarrow \mathcal{N}(\mu_{i-1}, \Sigma_{i-1})$
        $Y_i^j \leftarrow \text{Simulation}(X_i^j)$
        $E_i^j \leftarrow \text{Fitness}(X_i^j, Y_i^j)$
    end for
    $E_i = \min\{E_i^j\}_{1 \le j \le N_s}$
    $\Phi_i \leftarrow$ Choose N_b samples among $\{(X_i^j, Y_i^j)\}_{1 \le j \le N_s}$
    $(\mu_i, \Sigma_i) \leftarrow$ Update the distribution based on Φ_i
    if $\|E_i - E_{i-1}\| < \epsilon$ **then**
        Exit()
    end if
end for

We minimize the energy function by using a CMA algorithm [31]. Breifly speaking, the CMA is an iterative procedure that begins with initial parameter X_0 and takes random samples around X_0 with a multivariate normal distribution $\mathcal{N}(\mu_0, \Sigma_0)$, where $\mu_0 = X_0$ is its mean and Σ_0 is an initial covariance matrix (see Algorithm 3). The random samples are evaluated with respect to the fitness function to select a subset of best samples. The mean and covariance of the multivariate normal distribution is then updated based on the best subset. New samples for the next generation are selected from the updated normal distribution $\mathcal{N}(\mu_1, \Sigma_1)$ (see Figure 5.3). The algorithm repeats this procedure until the fitness converges.

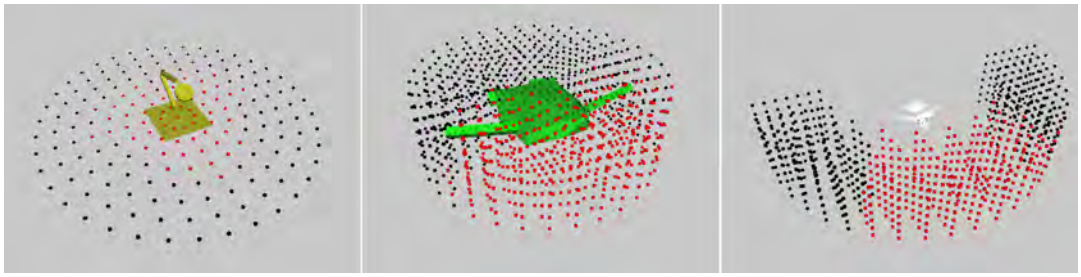


Figure 5.4 The grid of target locations. The target locations are sampled equidistantly regardless of the radius from the center. The grid point is red if the fitness energy is below a user-specified threshold and the controller is 5-cycle stable. In practice, a 5-cycle stable locomotion would repeat indefinitely while maintaining its balance. The turtle can maneuver reliably in the range of steering angles $[-60°, 60°]$ and elevation angles $[-65°, 65°]$. The bird can maneuver stably in the range of steering angles $[-30°, 30°]$ and elevation angles $[-35°, 35°]$.

5.2.3 Control Parameterization

The simulation of a dynamic model with its skill parameters $X \in R^N$ generates an outcome $Y \in R^M$ which is parameterized in task space. Conceptually, a parameterized motor skill is an inverse process that maps task paramters Y to skill paramters X. Usually, the dynamic system has more skill parameters than task parameters such that $N > M$. We construct the parameterized motor skill by example. Each example encodes what outcome Y_j is expected for an instance of skill parameters X_j. We begin with a user-provided base controller that performs the task Y_0 with the initial parameter X_0. The systematic application of control adaptation can generate a collection of simulation examples, (X_j, Y_j). The tasks Y_j are sampled on a grid of target locations around the synthetic creature. Given a series of task samples, we can adapt the base controller to find skill parameters for each task. This process is computationally demanding. We discuss an efficient construction method in the next section.

The controller decides which action to take at runtime based on a regression over a collection of simulation examples (X_j, Y_j). The regression allows the controller to

decide parameters X immediately for any given task Y without time-consuming optimization. The regression of skill-task parameters does not account for passive DOFs, which affect the outcome of the simulation. Accounting for passive DOFs can make the controller more accurate and reliable. We employ differential regression to deal with the deficiency of state observations [42]. Roughly speaking, differential regression infers the difference of skill parameters from the difference of task parameters. Let S_0 be a nominal reference state of the dynamic system. The base controller starting from S_0 performs the task Y_0 with the initial parameter X_0. At runtime, our controller runs a preparatory simulation with the nominal reference parameter X_0 to perform task Y', which may differ from Y_0 if the current state S including passive DOFs is different from its nominal reference S_0. The differential regression decides $\Delta Y = Y - Y'$ based on a collection of differential examples $(\Delta X_j, \Delta Y_j)$, where $\Delta X_j = X_j - X_0$ and $\Delta Y_j = Y_j - Y_0$. We employ k-NN (Nearest Neighbor) regression, which selects k nearest neighbors from the examples. The dissimilarity is measured by $d_i = \|\Delta Y - \Delta Y_i\|_{\mathbf{w}}$, where $\mathbf{w}$ is a weight vector. The output ΔX of the regression is computed as a weighted sum of its neighbors

$$\Delta X = \sum_{i \in k\mathrm{NN}} \frac{d_i^{-1}}{D} \Delta X_i, \tag{5.9}$$

where $D = \sum_{i \in k\mathrm{NN}} d_i^{-1}$. The weight of each example is inversely proportional to the dissimilarity from ΔY. The controller advances the simulation with skill parameters $X = \Delta X + X_0$.

Grid Construction The six-dimensional target parameters for our turtle and bird include both linear and angular displacements, which are strongly coordinated. For example, when the bird makes a turn, its body rolls towards the inside of the turn to compensate the centrifugal force. The bank (rolling) angle depends on the curvature of the turning trajectory. Exploiting this correlation, we construct a three-dimensional (instead of six-dimensional) grid of target locations in a cylindrical coordinate system (see Figure 5.4). The correlation between linear and angular displacements determines the target orientation at each grid location (see [42] for the calculation of the bank

angle). At each grid location, we can have a sub-grid of angular displacements around the computed target orientation to distribute samples over angular perturbation. The per-point sub-grids make the controller more resilient to external perturbation at extra computational cost. In our experiments, balance control is critical for bird flight, and thus our flying controller benefits substantially from the construction of angular sub-grids. On the other hand, the notion of dynamic balance does not apply to underwater swimming because we assumed zero gravity underwater. Our turtle maneuvers reliably without the angular sub-grids.

Dynamic Stability. The notion of dynamic balance/stability is important for terrestrial and aerial locomotion. Deciding whether an under-actuated skill is dynamically stable or not is not straightforward. We define the dynamic stability of a controller in a parametric manner. The controller with skill parameters X is 1-cycle stable if the controller starting at the initial state $S^{(0)}$ brings the system to a new state $S^{(1)}$ at the next cycle and the weighted norm of their displacement $\|S^{(1)} \ominus S^{(0)}\|_{\mathbf{w}} < \epsilon$ is below the user-specified threshold ϵ given a weight vector $\mathbf{w}$. Note that $\ominus$ denotes the difference between articulated poses. We use the optimal distance metric for articulated poses that effectively removes translation in the horizontal plane and rotation about the vertical axis [55]. We can also define n-cycle stability for any n. The controller is n-cycle stable if $\|S^{(i)} \ominus S^{(0)}\|_{\mathbf{w}} < \epsilon$ for any $0 < i \leq n$. In practice, 1-cycle stability does not provide much confidence as to how stable the controller would be. We can have stronger confidence with larger n (see Figure 5.4). In our experiments, 5-cycle stable controllers are selected and employed for interactive control of simulated creatures.

5.2.4 Efficient Construction

Forward dynamic simulation for evaluating each sample is the computational bottleneck. Control parameterization runs CMA optimization at a grid of N_p samples. The CMA optimization takes N_s simulation trials at each generation and iterates up to N_g generations. The construction of a parameterized skill requires $O(N_s \times N_g \times N_p)$

open-loop forward dynamic simulations. Reducing the number of simulation trials is the key to the efficient construction of parameterized motor skills.

A critical observation is that many of the simulation trials the CMA optimization generated for one target location can be reused for the next optimization with another target location. Control parameterization runs many CMA optimizations for control adaptation. The standard CMA algorithm is memoryless in the sense that it maintains only its current state and discards all previous computation results. Our recollective CMA algorithm considers a system of CMA optimizations. The algorithm caches the samples and their evaluation results during optimiztion and reuses cached samples, instead of taking new samples, to accelerate the overwhole process.

The recollective CMA algorithm is particularly useful if the evaluation of each individual sample is computationally expensive. In our case, the evaluation requires forward simulation of an articulated dynamic system and thus can be slow. However, storing the simulation result takes only a small amount of memory space. Precisely, the simulation data include skill parameters, the total torque, and the position, orientation, linear velocity, and angular velocity of the body at the end of the simulation. This information allows us to re-evaluate the fitness energy with respect to a different target location.

Algorithm 4 Recollective CMA algorithm

N_g : maxinum # of generations

N_s : # of samples in each generation

1: set μ_0 and Σ_0
2: **for** $i \leftarrow 1, N_g$ **do**
3: **for** $j \leftarrow 1, N'_s$ **do**
4: $X_i^j \leftarrow \mathcal{N}(\mu_{i-1}, \Sigma_{i-1})$
5: $Y_i^j \leftarrow \text{Simulation}(X_i^j)$
6: $E_i^j \leftarrow \text{Fitness}(X_i^j, Y_i^j)$
7: save (X_i^j, Y_i^j) into the cache
8: **end for**
9: Choose samples $\{(X_i^j, Y_i^j)\}_{N'_s < j \leq N_s}$ from the cache
10: $E_i = \min\{E_i^j\}_{1 \leq j \leq N_s}$
11: $\Phi_i \leftarrow$ Choose N_b samples among $\{(X_i^j, Y_i^j)\}_{1 \leq j \leq N_s}$
12: $(\mu_i, \Sigma_i) \leftarrow$ Update the distribution based on Φ_i
13: **if** $\|E_i - E_{i-1}\| < \epsilon$ **then**
14: Exit()
15: **end if**
16: **end for**

The benefit of reusing simulation trials is twofold. First, given a target location of the subsequent CMA optimization, we can choose an initial guess closer to its optimal solution based on previous trials. We re-evaluate the fitness of every trial to choose the best one as the initial guess. Secondly and more importantly, reusing cache samples can reduce the number of simulation trials at each generation of CMA optimization. The algorithm overview is as follows (see Algorithm 2). We choose a small number of new samples at each generation with an evolving multivariate normal distribution suggested by CMA (line 3–8). Additional samples chosen from the cache supplement

the new samples to enrich the distribution (line 9). While evaluating the fitness of a new sample necessitates running an open-loop simulation (line 6), we do not need to re-run the simulation for cache samples. The computational cost for searching and adding cache samples is almost negligible comparing to the simulation costs for new samples.

More specifically, the new samples $\{X_i | i = 1, \cdots, N'_s\}$ in each generation perform tasks $\{Y_i | 1, \cdots, N'_s\}$. We would like to choose supplementary samples $\{(X_i, Y_i) | i = N'_s + 1, \cdots, N_s\}$ from the cache that perform well with respect to the intented task. To do so, we consider the mean and covariance of $\{Y_i\}$ in the task space,

$$\mu_Y = \frac{1}{N'_s} \sum_{i=1}^{N'_s} Y_i \tag{5.10}$$

$$\big(\Sigma_Y\big)_{kl} = \frac{1}{N'_s} \sum_{i=1}^{N'_s} \big((Y_i)_k - (\mu_Y)_k\big)\big((Y_i)_l - (\mu_Y)_l\big) \tag{5.11}$$

and choose cache samples that fall into the range $(Y - \mu_Y)^\top \Sigma_Y (Y - \mu_Y) < c$. In our experiments, $c = 1$ unless mentioned otherwise. Selecting cache samples in the task space guides the CMA optimization quickly towards the target location. We employed a kd tree implementation, which allows incremental tree updates, to search cache samples efficiently [79].

5.2.5 Experimental Results

The dynamics of our creatures are simulated on Virtual Physics, which is an open source library for simulating rigid and articulated body dynamics [43]. The parameters and coefficients for physics simulation and CMA optimizations are summarized in Figure 5.5. We employed a simplified fluid dynamics model to simulate aerodynamics of bird flight and hydrodynamics of turtle swimming [116]. The regression-based simulation and control is fast enough to allow for interactive control of creature locomotion, swimming, and flying.

Performance Comparison. We compare the performance of our construction method to alternative techniques (see Figure 5.6). The blue graph labeled "nearest"

Model	Luxo		Turtle		Bird	
PD gains	P	17000 (base1)	P	10000 (shoulder)	P	20 (shoulder)
		10000 (base2)				9 (elbow1)
		6000 (leg)				5 (elbow2)
						5 (wrist)
	D	0.01 (base1)	D	110 (shoulder)	D	0.004 (shoulder)
		0.005 (base2)				0.004 (elbow1)
		0.004 (leg)				0.001 (elbow2)
						0.001 (wrist)
Gravity (N/kg)	9.81		0		9.81	
Density (kg/m³)			1000 (water)		1.225 (air)	
# of max generations	60		60		50	
# of samples at each generation	30		30		50	

Figure 5.5 Parameters and coefficients for the physics simulation and CMA optimization.

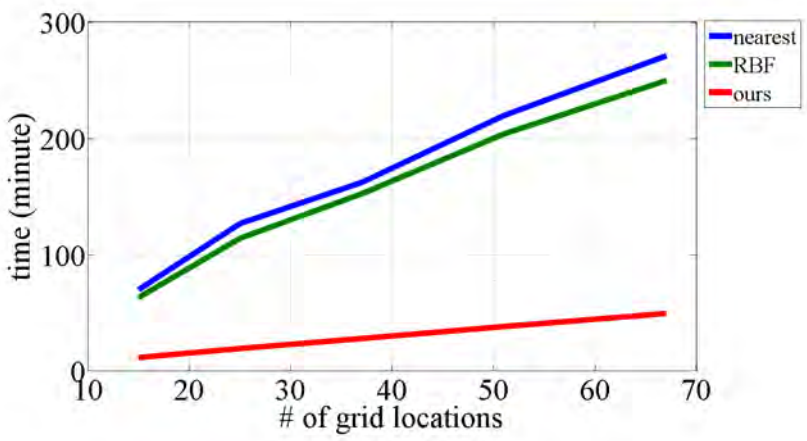


Figure 5.6 Performance comparison with the construction of a motor skill of the luxo.

is the performance of the standard CMA that chooses the initial configuration from the nearest grid point with previously optimized results, following the spirit of the continuation method [121]. The green graph labeled "RBF" is also the performance of the standard CMA that chooses the initial configuration by extrapolating previous results, as discussed by Liu et al. [68]. The graph in Figure 5.6 shows that the use

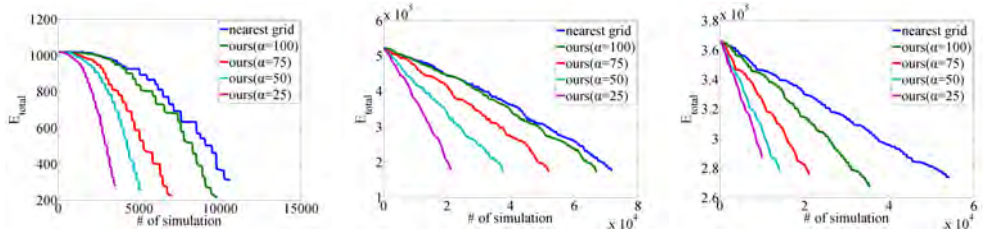


Figure 5.7 Convergence of one-by-one optimization. α is the percentage of new samples in each generation.

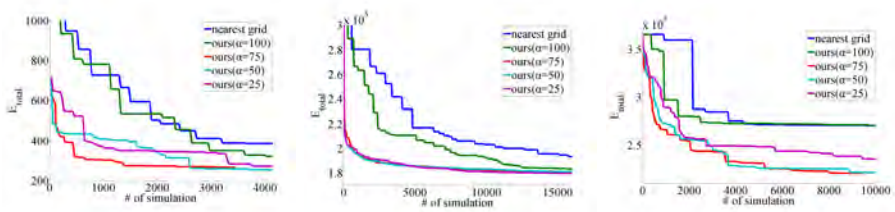


Figure 5.8 Convergence of simultaneous optimization.

of RBF interpolants achieves small performance gain from the nearest method. Our method achieves far greater performance gain (six times faster) over two alternative methods. The computation time of the nearest and RBF methods, respectively, increases linearly with the number of grid locations, while the computation time of our method increases sub-linearly if grid locations are distributed coherently. Sub-linear time complexity can be realized, because our method cumulates the results of simulation trials. Optimization with a large collection of trial data can be faster than optimization without any prior knowledge.

Performance Improvement. The total fitness function $E_{total} = \sum_i E_i$ measures the convergence of a system of per-grid optimizations, where E_i is a fitness value at i-th target location (see Figure 5.7). We initialized fitness values with the initial skill parameters and incrementally updated the values by optimizing skills at each grid location one by one. We used 48, 125 and 442 target locations for the luxo, the turtle, and the bird, respectively, in this experiment. The graphs show that the more cache samples we use, the faster the optimization converges. The maximal use of cache

samples depends on the dimension $N_t = \dim(Y)$ of the task space. The number of new samples at each generation should be large enough to expect a nondegenerate task-space normal distribution in which cache samples are chosen. CMA updates the normal distribution in the skill space based on the best subset of samples to proceed to the next generation. When we mix 25% of new samples and 75% of cache samples, 47.2% of the best subset at each generation are cache samples on average. This means that caching samples practically affects the performance and convergence of CMA optimization.

Convergence Analysis. CMA is a stochastic process and therefore the standard CMA would converge to different solutions with different random seeds. Interestingly, our method converges better (lower average fitness) and more consistently (lower average variance) than the standard CMA method, regardless of the reusing percentage of cache samples. For better visualization of the convergence tendency, we show the convergence graphs of simulataneous optimization in Figure 5.8. Unlike the graphs in Figure 5.7, simultaneous optimization updates the fitness values of all unvisited target locations after visiting each individual target location. The fitness values at unvisited targets might improve as new samples are cached and thus potentially better initial configurations can be suggested from the cache. Figure 5.8 shows that our recollective CMA converges faster than the standard CMA algorithm and suggests better solution (lower total fitness energy) after convergence. The contribution of cache samples can be examined more specifically. If we use cached samples only for suggesting initial configurations ($\alpha = 100\%$), the rate of convergence would improve from the standard CMA ("nearest grid"), but would eventually converge at similarly-rated results. The use of supplementary cache samples ($\alpha = 75\%$, $\alpha = 50\%$, or $\alpha = 25\%$) achieves not only faster convergence rates, but also convergence at better solutions in all three examples. This observation is particularly noticeable for the bird example, which poses challenging energy landscapes with numerous local extrema (see Figure 5.8 right). The prior knowledge based on the cache samples allows our recollective CMA algorithm to avoid local extrema more effectively than the memoryless standard CMA

algorithm. Faster convergence rates suggest further performance gain (by factor of 5 to 20 times) over the improvement we mentioned previously. The performance gain depends on the convergence threshold. Stricter thresholds tend to make our algorithm even faster than the standard CMA algorithm. The optimization with $\alpha = 50\%$ or $\alpha = 75\%$ sometimes outperforms the optimization with $\alpha = 25\%$ taking the convergence rate into account. The best ratio may be determined by the characteristics of each individual problem.

Maneuverability vs Grid Density. The accuracy of control depends on the range and density of grid locations. We depicted the relation between maneuverability and grid density based on the simulation of the turtle (see Figure 5.9). The test trajectory was designed to have both turns and height variations, resembling an 8-shape from the top view and twisted from the front view. We evaluated how well the turtle maneuvers following the test trajectory. Three evaluation measures were used. $E_p = \sum_i \|p_i^d - p_i\|$ and $E_q = \sum_i \|\log(q_i^{-1} q_i^d)\|$, respectively, measure how accurately the turtle tracks target locations and orientations. $E_o = \sum_i \|\log(q_i^{-1} q_{i+1})\|$ measures how smoothly the turtle swims without oscillation. It was found that the turtle maneuvers robustly with only 30 to 60 samples for differential regression.

Underwater Navigation. We built a virtual underwater environment, in which the user can control our turtle interactively by selecting its target locations (see Figure 5.1 and the accompanied video). The motor skill uses 120 samples for parameterization and differential regression. Constructing the motor skill took about 30 minutes on a modern desktop CPU. The turtle maneuvers responsively to swim in arbitrary directions and make quick turns. There is also a seafloor vent that generates upward flow of seawater. The turtle is pushed strongly above the vent and quickly recovers its control.

Interactive Bird Control and Retargeting. The motor skill of the bird uses 707 samples (101 grid locations and a sub-grid of 7 angular perturbations per each location) for parameterization. The construction took about 17 hours. The computation was performed mostly on a single CPU core except for the simulation of deformable

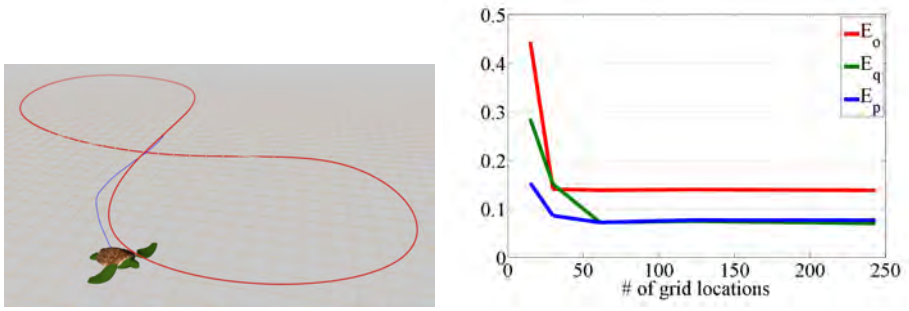


Figure 5.9 Evaluation of maneuverability with the density of grid locations.

feathers, which utilized three cores of a four-core desktop CPU. The generalization capability of our approach allows us to learn motor skills of an imaginary or extinct creature, such as *Archaeopteryx*. Note that data-driven approaches, such as the work by Ju et al. [42], cannot construct controllers for either imaginary or extinct animals because they require motion capture data for training. We created an imaginary bird with extraordinarily long feathers, like a peacock (see Figure 5.1). The primary and secondary feathers are one and a half times longer than the original dove feathers. The tail feathers are ten times longer and 80 times stiffer. The long, whippy feathers make flight control even more challenging. The optimization of the base controller makes the wings beat slower to match the natural frequency of longer feathers. The imaginary bird flies in realtime, and its fully-optimized motor skill is as robust and maneuverable as the motor skill of the original dove.

5.2.6 Discussion

We presented an optimization-based method that constructs a parameterized controller automatically given a base controller, which serves as a seed of exploration in the control space. Our construction algorithm generates variations of a base control while preserving its gait and characteristics as much as possible. Stylistic and expressive locomotion can also be generated if appropriate base controllers are provided.

In this work, we use a simple form of base controllers that do not generate any feedback or feedforward signal. Our algorithm can benefit from the use of a more

sophisticated form of base controllers. For example, recent biped controllers equipped with feedback mechanisms can maintain balance actively. Parameterizing such a self-balancing controller would require fewer, sparser samples for regression than parameterizing a simple base controller. A similar observation can be found in the comparison between the turtle and the bird. The turtle does not require balance control and, therefore, can be controlled using a small number of samples. The controller of the bird uses many more samples to provide precise control over the three-dimensional pose in flight.

Our regression-based controller does not construct an explicit model of control, but evaluates control responses at runtime in a lazy manner. Lazy evaluation is advantageous if the underlying model is complex. For example, the wingbeats of our bird model forms a 29-dimensional space and only a tiny fraction of wingbeats in the space are either natural-looking or functionally-useful. Natural wingbeats form a low-dimensional sub-manifold in the high-dimensional wingbeat space. Ideally, we wish to have an explicit mathematical model that fits the sub-manifold for better runtime performance. However, as shown in previous work [42], the sub-manifold is highly non-linear and necessitates a complex non-linear model to fit. The flexibility and versatility of lazy evaluation allows us to deal with arbitrary morphologies and simulation environments.

Our algorithm generates control samples via optimization. Alternatively, a collection of motion data captured from live subjects can serve as training data for our regression-based controller [102, 42]. The data-driven control approach has many advantages, though it is burdened with the difficulty of data acquisition. Motion capture references reproduce the natural motion of creatures. The data-driven algorithms are usually simply, easy-to-implement, and computationally efficient because motion capture references provide prior knowledge on the control problem. On the other hand, the optimization-based approach is flexibly, versatile, and generalizes easily. The advantages and disadvantages of the data-driven approach is reciprocal to those of the optimization-based approach. The data-driven and the optimization-based ap-

proaches are at two extremes of a spectrum. We anticipate that new approaches would emerge from the middle of the spectrum to take the advantages of both approaches simultaneously.

5.3 Example-Guided Control by Deep Reinforcement Learning

Reinforcement Learning (RL) for generating adaptive motor skills has attracted great attention. Given the current state of a character, the character is provided with a set of actions (or a continuous spectrum of actions) to choose from. The control policy (or controller) decides the best action to take, which results in a subsequent state and a reward associated with the state transition. The goal of reinforcement learning is to find an optimal policy, which maximizes the sum of expected future rewards.

Recently, Deep Reinforcement Learning (DRL), which combines reinforcement learning with deep neural networks, has demonstrated its potential in physics-based control and simulation. It is not obvious if deep reinforcement learning would generalize to deal with professional-quality characters with many degrees of freedom, given that we do not have any training data for the motion of imaginary creatures. The motor skill is previously defined by leveraging immediate rewards of taking actions at each state. Specifying rewards can be cumbersome from the viewpoint of animators and practitioners, who might want to have direct control over the motor skills. It is furthermore not obvious how to define a diversity of motor skills in the framework of reinforcement learning.

We present a control method for flying creatures, which are simulated aerodynamically in three-dimensional virtual environments, controlled interactively, and equipped with a variety of motor skills. Each motor skill is represented as deep neural networks, which construct a mapping from observed states to joint actions with continuous control. The user may provide a sequence of keyframes, from which our system constructs a control policy for steering the flying direction and performing a specific motor skill.

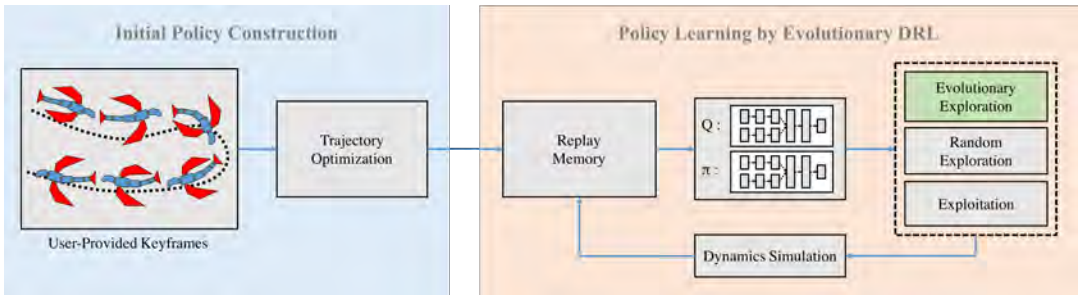


Figure 5.10 System Overview.

The challenge is that deep reinforcement learning often converges to local minima with high-dimensional dynamical systems. The convergence of learning is closely related to both policy update methods and exploration strategies. Our learning algorithm is inspired by Covariance Matrix Adaptation Evolution Strategy (CMA-ES), which is a stochastic, derivative-free method for numerical optimization of non-linear or non-convex continuous optimization problems. Our novel algorithm combines the smart exploration strategy of CMA-ES with the generality of Deep Q-Learning (DQL), which is one of DRL methods, to improve the convergence rate and the final quality of the control policy substantially. The effectiveness of our *Evolutionary DQL* method will be demonstrated with imaginary winged creatures and their motor skills learned automatically from user-provided keyframes.

5.3.1 System Overview

An illustrative overview of our framework is shown in Figure 5.10. The input to our system is keyframe animation $\{\mathbf{q}_1, \cdots, \mathbf{q}_T\}$ of a creature specified by novice users or professional animators, where $\mathbf{q}_t$ is the pose at time t and T is the length of the animation. The output of the system is a control policy $\pi(\mathbf{s})$ that maneuvers the creature in a physically plausible manner. Here, $\mathbf{s}$ is the sensorimotor state of the creature, which includes its dynamic state (i.e., joint angles and its velocities) and the environment state (e.g., the location of targets and obstacles). The environment state is task-dependent, so defined differently for each individual task and control

policy.

The system dynamics $p(\mathbf{s}_t, \mathbf{a}_t)$ leads to the next state $\mathbf{s}_{t+1}$, given current state $\mathbf{s}_t$ and action $\mathbf{a}_t$. Our winged creature is composed of rigid bones connected by joints and thin shells attached to the bones (see Figure 5.11). We assume that aerodynamic forces act only on the thin shells and then transferred to the articulated body system of dynamics. The simplified aerodynamics model from previous studies [117, 42] generates drag and lift forces acting on individual thin shells. The drag force $\mathbf{f}_d$ simulates resistance to the air and the lift force $\mathbf{f}_l$ simulates the airfoil effect generated by the air pressure difference between upside and downside of the wing. The forces are computed for each individual polygon of the thin shells.

$$\mathbf{f}_d = \frac{1}{2}\rho \mathbf{v}^2 A C_d \qquad \text{and} \qquad \mathbf{f}_l = \frac{1}{2}\rho \mathbf{v}^2 A C_l \tag{5.12}$$

where $\mathbf{v}$ is the relative velocity to the air and A is the area of the polygon. C_l and C_d are lift and drag coefficients that vary in accordance with the *angle of attack* of the polygon.

Action $\mathbf{a}$ in our system is represented as a target pose that the creature follows during the next time step. Proportional Derivative (PD) servos are used to generate torques at actuated joints to track the target pose, while the root of the skeleton and passive joints remain unactuated. Although we can represent action as torques acting at joints directly, the use of PD servos achieves better motor skills in our experiments. Similar observations can also be found in previous studies [76, 86].

The keyframe animation provided by the user usually includes several wingbeats that propel the creature straight ahead. In practice, the creature simulated with the keyframe animation will loose its balance and fall down immediately, because handcrafted animation is almost always physically implausible. The trajectory optimization module in Figure 5.10 transforms the user-provided animation into a physically plausible one, which can be reproduced by open-loop forward dynamics simulation with aerodynamic forces. The animation thus obtained provides us with a collection of experience tuples $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$, which record successful execution of actions. The

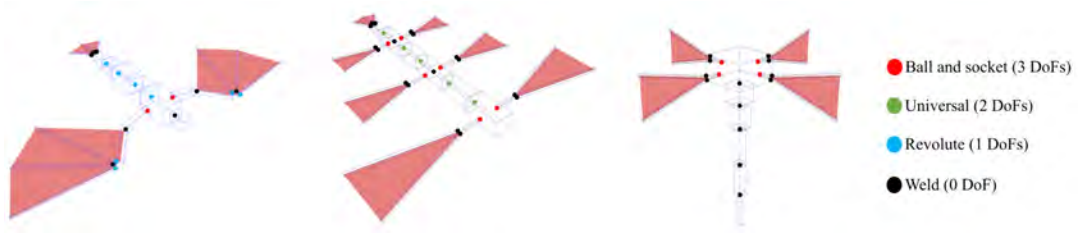


Figure 5.11 The dynamic models of our creatures: (from left to right) Dragon, six-winged worm, and four-winged spore.

experience tuples serve as initial training data for control policy learning.

Reinforcement learning makes use of both exploitation and exploration strategies. The exploitation strategy develops the estimation of future rewards by making the best use of the information currently available, while the exploration strategy probes unseen states and actions to gather more information to improve the control policy. Although our initial training data have very limited information on flying straight ahead, the exploration strategy perturbs the initial policy to learn how to turn, soar, glide, and dive. During the learning process, finding a good balance between exploitation and exploration is essential. If exploitation is dominant, the learned policy tends to be suboptimal because it could not fully observe better choices. If exploration is dominant, newly-gathered information continuously changes the estimation of future rewards before it develops the true values and thus hinders the policy from convergence.

We utilize Deep Q-Learning (DQL) as a basis [75, 64], which is a variant of reinforcement learning that represents both state-action value functions (a.k.a. Q-value functions) and policy functions as deep neural networks. In theory, DQL is able to handle control systems with high-dimensional sensory inputs and control outputs. However, achieving stable convergence is very challenging in practice because of many reasons. We identified two key observations to improve the convergence of learning with high-dimensional dynamic systems. First, the robustness of the initial policy generated by trajectory optimization is critical to the convergence of learning. Tra-

jectory optimization typically defines an energy function and constructs an output trajectory that minimizes the energy function. As pointed out by Wang et al [111], this optimal-energy trajectory could be fragile even for small perturbation since the simulated creature would put barely sufficient effort to actuate its joints, but not sufficient to resist against unexpected perturbation. This frailty of the optimal trajectory hinders exploration in RL because random perturbation of the initial policy would fail very frequently. In the next section, we will discuss how to make the optimized trajectory robust against random perturbation by employing the idea of stochastic optimal control.

Secondly, repeatedly perturbing straight ahead flight with random noise would eventually discover new motor skills such as soaring and hovering if unlimited computation resources are provided. However, exploration by such a random strategy would be extremely slow and thus policy learning can benefit from smarter exploration strategies. We found that the evolutionary strategy of CMA-ES supplements the random strategy to explore unseen states and actions rapidly and stably. Unlike previous studies that alternate between trajectory optimization and direct policy learning [60, 76], we plug the evolutionary strategy directly into DQL as its part, so we can make use of intermediate samples in evolution as well as the final optimized result. This type of fusion is feasible only with sampling-based optimization such as CMA-ES and achieves substantial speedup and better convergence over the standard DQL. We will explain the use of CMA-ES for stochastic trajectory optimization in Section 4, followed by our *Evolutionary DQL* in Section 5 that takes the key component of CMA-ES for improving DQL.

5.3.2 Initial Policy Construction

Given user-provided keyframe animation, the primary goal of trajectory optimization is to transform it into physically valid animation while maximizing rewards. Physically valid animation can be represented as a sequence of actions $\{\mathbf{a}_1, \cdots, \mathbf{a}_T\}$ and initial state $\mathbf{s}_0$. Forward dynamics simulation applies system dynamics recursively to

generate a simulated trajectory $\{\mathbf{s}_0, \cdots, \mathbf{s}_T\}$ such that

$$\mathbf{s}_i = p(\mathbf{s}_{i-1}, \mathbf{a}_i) \tag{5.13}$$

for $i = 1, \cdots, T$. The secondary goal is to have the simulated trajectory resilient against random perturbation such that the deviation of the simulated trajectory is bounded within a certain range when random noises are added to the actions.

Rewards. The reward function is a cumulative sum of immediate rewards over the duration of the animation,

$$r(A, S) = \sum_{i=1}^{T} r(\mathbf{s}_{i-1}, \mathbf{a}_i, \mathbf{s}_i), \tag{5.14}$$

where A is a sequence of actions $\{\mathbf{a}_1, \cdots, \mathbf{a}_T\}$ and S is a sequence of the resultant states $\{\mathbf{s}_0, \cdots, \mathbf{s}_T\}$. The immediate reward function involves five terms.

$$r = r_{\text{target}} + r_{\text{collision}} + r_{\text{effort}} + r_{\text{balance}} + r_{\text{regul}} \tag{5.15}$$

The first term r_{target} drives the creature towards its target position.

$$r_{\text{target}} = -w_1 \|\mathbf{p}\|^2, \tag{5.16}$$

where $\mathbf{p}$ is the position of the target with respect to the body local coordinate system. $r_{\text{collision}}$ penalizes interpenetration through obstacles.

$$r_{\text{collision}} = \begin{cases} -w_2 d^2, & \text{if collision occurs} \\ 0, & \text{otherwise} \end{cases} \tag{5.17}$$

where d is the penetration depth. The third term r_{effort} minimizes the effort of flying.

$$r_{\text{effort}} = -w_3 \|\tau\|^2, \tag{5.18}$$

where τ is the vector of torques generated by PD servos. The fourth term r_{balance} penalizes abrupt rotational movements to keep the creature balanced during flight.

$$r_{\text{balance}} = -w_4 \|\omega\|^2 - w_5 \|1.0 - \mathbf{v} \cdot \mathbf{u}\|^2, \tag{5.19}$$

where $\mathbf{v}$ and $\mathbf{u}$ are the up-vectors of the skeleton root and the world, respectively, and ω is the angular velocity at the root. r_{regul} is a regularization term that prevents large deviations from the user-provided keyframe animation.

$$r_{\text{regul}} = -w_6 \sum \|\mathbf{a}_i - \mathbf{q}_i\|^2, \tag{5.20}$$

where $\mathbf{q}_i$ are frames of the user-provided animation.

Trajectory Optimization. The primary goal can be achieved by formulating the problem as nonlinear optimization taking actions as parameters to be optimized. We employ CMA-ES for the optimization, which is usually formulated to minimize a certain energy function. Conversely, CMA-ES maximizes rewards according to the convention of RL and shares the same reward function with RL in the next section. CMA-ES is an iterative procedure that begins with initial action sequence $A_0 = \{\mathbf{a}_1, \cdots, \mathbf{a}_T\}$. The whole action sequence over duration $[1, T]$ is treated as a single point in a high-dimensional space. The optimization procedure takes a collection of random action sequences over the same duration by perturbing the initial one with a multivariate normal distribution, where μ_0 is its mean and Σ_0 is the initial covariance matrix (see Algorithm 5, line 4). Each action sequence A_j over the duration generates a simulated trajectory S_j via forward dynamics simulation (line 5). The resultant trajectories are evaluated with respect to the reward function to select a subset of best trajectories (line 6-8). The mean and covariance of the multivariate normal distribution are then updated based on the subset (line 9). New action sequences for the next generation will be selected from the updated normal distribution. The algorithm repeats this procedure until the reward converges (line 10-11). The user-provided keyframe animation driven by PD servos serves as the initial action sequence, which transforms gradually as the iteration proceeds.

Algorithm 5 Covariance Matrix Adaptation Evolution Strategy

μ_0 : initial mean

Σ_0 : initial covariance

N_g : maximum # of generations

N_s : # of trajectories generated in each generation

1: **procedure** CMA-ES
2: **for** $i = 1, \cdots, N_g$ **do**
3: **for** $j = 1, \cdots, N_s$ **do**
4: $A_j \leftarrow \mathcal{N}(\mu_{i-1}, \Sigma_{i-1})$
5: $S_j \leftarrow p(s_0, A_j)$
6: $r_j \leftarrow \mathrm{r}(A_j, S_j)$
7: **end for**
8: $R_i = \max\{r_1, \cdots, r_{N_s}\}$
9: $\Phi_i \leftarrow$ Choose N_b best actions among $\{\mathbf{A}_1, \cdots, \mathbf{A}_{N_s}\}$
10: $(\mu_i, \Sigma_i) \leftarrow$ Update the distribution by Φ_i
11: **if** $\|R_i - R_{i-1}\| < \epsilon$ **then**
12: Break
13: **end if**
14: **end for**
15: **end procedure**

Optimization under uncertainty. Achieving the secondary goal requires further modification of the optimization procedure to account for the stability of each individual action sequence at line 6 of the algorithm. If the action sequence is stable, its reward value would not vary significantly at the present of small perturbation. Therefore, improving the stability of A_j entails maximizing the expected rewards at the neighbor of A_j. To account for the modification, line 6 is replaced with a procedure that we draw random actions $\{A'_k\}$ from $\mathcal{N}(A_j, \mathrm{diag}(\sigma))$ and then compute the expected reward of A_j as $r_j = E[r(A'_k, S'_k)]$. σ is related to the robustness expected for

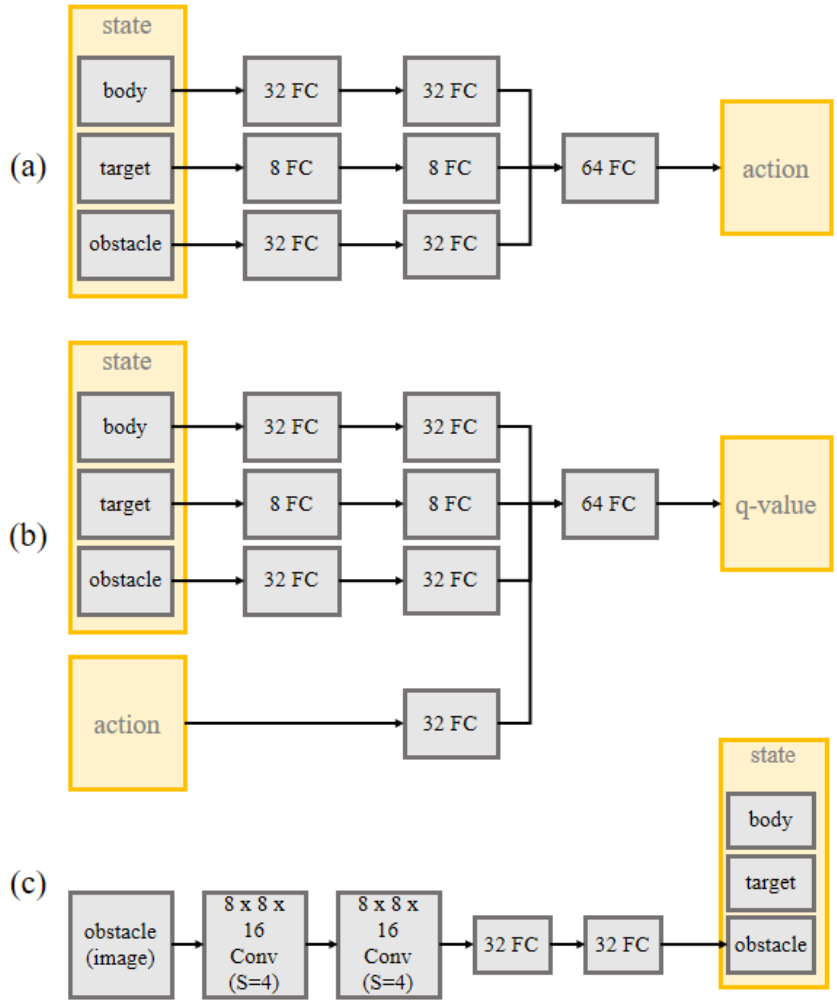


Figure 5.12 The structure of Neural Networks. (a) The policy network, (b) the state-action value (Q) network, and (c) convolutional layers for visumotor control. Noth that the convolutional layers are plugged to the sensory inputs of the policy and Q networks.

the resulting trajectory. Larger value of σ tends to make the trajectory more robust, though excessively large values of σ would make it converge with difficulty.

5.3.3 Evolutionary Deep Q-Learning

The initial control policy provides us with very little information considering the enormity of the state and action spaces. We now discuss how to generalize the information such that the learned policy can respond well at arbitrary situations with continuous control.

The reinforcement learning is defined by an agent and an environment. Given state $\mathbf{s} \in \mathbb{S}$ of an agent, action $\mathbf{a} \in \mathbb{A}$ is chosen by its control policy $\pi : \mathbb{S} \to \mathbb{A}$. Execution of the action brings the agent to a new state $\mathbf{s}' \in \mathbb{S}$ and the agent receives a reward $r : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to \mathbb{R}$ according to the desirability of the state transitioning. The goal of reinforcement learning is to find the optimal policy that maximizes the sum of expected future rewards $\mathcal{R} = \sum_{i=0}^{\infty} \gamma^i r_i$, where discount factor $\gamma \in [0, 1)$ is introduced to prevent the infinite sum from diverging. If we can predict the future rewards at any state, then determining the optimal policy becomes a simple matter. We can simply compare the sum of the immediate reward and its future rewards for all possible actions, then select the action that maximizes the sum. This idea is called Bellman backup. An approximator of future rewards $V(\mathbf{s})$ at state $\mathbf{s}$ is called a state value function. If $V(\mathbf{s})$ stands for the true value, its recursive relation can be written as follows.

$$V(\mathbf{s}) = \max_{\mathbf{a} \in A} (r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma V(\mathbf{s}')), \tag{5.21}$$

where A is a set of all possible actions at the state $\mathbf{s}$. We can construct the optimal state value function by iteratively replacing the left-hand side value with the right-hand side value for all states. Similarly, we can define a state-action value function that approximates the future reward at any state-action pairs.

$$Q(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma \max_{\mathbf{a}' \in A'} Q(\mathbf{s}', \mathbf{a}'), \tag{5.22}$$

where A' is a set of all possible actions at the subsequent state $\mathbf{s}'$. The benefit of this formulation is that we can directly determine the optimal policy as follows.

$$\pi(\mathbf{s}) = \operatorname*{argmax}_{\mathbf{a} \in A} Q(\mathbf{s}, \mathbf{a}). \tag{5.23}$$

Note that simulating the system dynamics for all possible actions is necessary with Equation 5.21, but not with Equation 5.23. A reinforcement learning method based on Equation 5.22 and 5.23 is called *Q-learning*.

Representation

The state and action of our system have continuous values. For continuous value problems in Equation 5.23, finding the optimal policy from the state-action value function requires yet another optimization in the action domain. To prevent this, we maintain a policy function $\pi(\mathbf{s}) : \mathbb{S} \rightarrow \mathbb{A}$ and a state-action value function $Q(\mathbf{s}, \mathbf{a}) : \mathbb{S} \times \mathbb{A} \rightarrow \mathbb{R}$ separately, approximating both as deep neural networks with weights θ_π, θ_Q, respectively [99, 64]. Both networks have the same structure of three fully connected layers with tanh units (see Figure 5.12). Two separate layers at the front learn the features of dynamic states and environment/task states independently. The two layers are combined and then followed by another fully connected layers. The size of output layers varies according to the dimension of $\pi(\mathbf{s})$ and $Q(\mathbf{s}, \mathbf{a})$. Each network has approximately 10k free parameters.

Evolutionary Exploration

Our learning method is episodic and trajectory-centric. In each episode, our creature is provided with a new goal (e.g. a target position) and performs a sequence of actions to achieve the goal. It may or may not achieve the goal successfully. While doing so, our creature generates a simulated trajectory from the reference starting position to the target position. We collect experience tuples from the trajectory to update the weight values of the Q and policy networks.

The key to successful learning in high-dimensional RL problems is to collect appropriate experience tuples by exploitation and exploration. The exploitation strategy performs actions according to the current policy without any perturbation, which can be understood as refining the future reward function more precisely by utilizing the known knowledge (see Algorithm 6, line 1–6). The exploration strategy performs

actions with perturbation in search for better policies, which can be understood as expanding the knowledge by gathering more information (see Algorithm 6, line 7–12). In continuous action spaces, adding random noise (usually Gaussian) around the current policy is a standard method for exploration because of its simplicity. However, this could make the policy remain in poor local minima especially in high-dimensional state/action spaces due to the curse of dimensionality.

The exploitation and random exploration strategies would fail to generate good trajectories in most episodes in the early phase of learning, because the control policy has not been trained yet. The RL methods are capable of learning even from failing experiences to some extent. So the control policy can make progress with a few successful episodes and many unsuccessful episodes although the learning rate will be rather slow. We found that the evolution strategy of CMA-ES can be used seamlessly for better exploration (see Algorithm 6, line 13–21). CMA-ES is a very powerful optimization method, which performs well in most episodes regardless of the progress of policy learning. Learning from success would be much faster than learning from failure. The evolutionary strategy explores the state-action space rapidly with far-reaching episodic goals.

One might worry about the performance of the evolutionary strategy in the RL framework because CMA-ES itself is computationally demanding. In the process of evolutionary optimization, a number of episodic trajectories are generated via forward dynamics simulation at each generation and thrown away afterwards. Fortunately, all the computation is not wasted, but can be reused for policy learning. We collect experience tuples from all intermediate trajectories as well as the final optimal trajectory. Therefore, a single episode using evolutionary exploration generates many coherent experiences, which provide momentum to steer the control policy in a desired direction. We will demonstrate in the experimental results that learning with evolutionary exploration converges faster and ends up with better control policies.

Algorithm 6 Exploitation and Exploration

procedure EXPLOITATION
  $E = \emptyset$
  for $i = 1, \cdots, T$ **do**
    $\mathbf{a}_i \leftarrow \pi(\mathbf{s}_i|\theta_\pi)$
    $\mathbf{s}_{i+1} \leftarrow p(\mathbf{s}_i, \mathbf{a}_i)$
    $E \leftarrow E \cup \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_{i+1})\}$
  end for
end procedure
procedure RANDOMEXPLORATION
  $E = \emptyset$
  for $i = 1, \cdots, T$ **do**
    $\mathbf{a}_i \leftarrow \pi(\mathbf{s}_i|\theta_\pi) + \mathcal{N}(\mathbf{0}, \text{diag}(\sigma))$
    $\mathbf{s}_{i+1} \leftarrow p(\mathbf{s}_i, \mathbf{a}_i)$
    $E \leftarrow E \cup \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_{i+1})\}$
  end for
end procedure
procedure EVOEXPLORATION
  $E = \emptyset$
  for $i = 1, \cdots, N_g$ **do**
    for $j = 1, \cdots, N_s$ **do**
      $\{\mathbf{a}_1, \cdots, \mathbf{a}_{N_s}\} \leftarrow \mathcal{N}(\mu_{i-1}, \Sigma_{i-1})$
      for $k = 1, \cdots, T$ **do**
        $\mathbf{s}_{k+1} \leftarrow p(\mathbf{s}_k, \mathbf{a}_k)$
        $E \leftarrow E \cup \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_{i+1})\}$
      end for
    end for
    $(\mu_i, \Sigma_i) \leftarrow$ Update the distribution by the elite set
  end for
end procedure

Algorithm 7 Learning

$Q|_{\theta_Q}$: state-action value network

$\pi|_{\theta_\pi}$: policy network

B : experience replay memory

repeat
  The environment/task states are initialized randomly
  $\rho \leftarrow U(0, 1)$
  if $\rho \leq \rho_0$ **then**
    $E \leftarrow$ EvoExploration
  else if $\rho \leq \rho_1$ **then**
    $E \leftarrow$ RandomExploration
  else
    $E \leftarrow$ Exploitation
  end if
  Put E into the replay memory B

  $X_Q, Y_Q \leftarrow \emptyset$
  $X_\pi, Y_\pi \leftarrow \emptyset$
  for $i = 1, \cdots, N$ **do**
    Sample an experience $\mathbf{e}_i = (\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)$ from B
    $y_i \leftarrow r_i + \gamma Q(\mathbf{s}'_i, \pi(\mathbf{s}'_i|\theta_\pi)|\theta_Q)$
    $X_Q \leftarrow X_Q \cup \{(\mathbf{s}_i, \mathbf{a}_i)\}$
    $Y_Q \leftarrow Y_Q \cup \{y_i\}$
    if $y_i - Q(\mathbf{s}_i, \pi(\mathbf{s}_i|\theta_\pi)|\theta_Q) > 0$ **then**
      $X_\pi \leftarrow X_\pi \cup \{\mathbf{s}_i\}$
      $Y_\pi \leftarrow Y_\pi \cup \{\mathbf{a}_i\}$
    end if
  end for
  Update Q by (X_Q, Y_Q)
  Update π by (X_π, Y_π)
until no improvement on the policy

Learning Algorithm

The learning algorithm begins with two separate networks $\pi(\mathbf{s})|_{\theta_\pi}$, $Q(\mathbf{s},\mathbf{a})|_{\theta_Q}$ with weights θ_s, θ_Q (see Algorithm 7). The weights are set by Xavier initialization, which is a fully automatic method for the network initialization [24]. The algorithm generates a new episode to gather experience tuples (line 2–10) and updates the networks using training data thus collected (line 11–22).

For each episode, a new task and a new environment are set randomly and then we choose a strategy probabilistically among exploitation, random exploration, and evolutionary exploration to address the task. Probabilities by ρ_0 and ρ_1 are decided such that the ratio of contribution of exploitation, random exploration, and evolutionary exploration becomes 3:1:1. This ratio was set by experimentation to prompt stable convergence of learning, while allowing conservative and rapid strategies to explore unseen states and actions in a balanced manner. Sometimes, we need to subsample experiences from evolutionary exploration, which may generate too many episodic trajectories in the evolutionary process depending on its parameter setting (e.g., the number of samples in each generation and the maximum of generations). The subsampling is a stochastic process, wherein highly-rewarded trajectories are more likely to be selected using a Boltzman distribution.

All the experiences are stored in a replay memory, which plays an important role in training the networks unbiasedly (line 10). When training the networks, a mini-batch of random samples from the replay memory are used instead of most recent experiences in order to break the temporal correlations by mixing recent and past experiences [75]. Otherwise, the bulk of subsequent experiences makes it difficult for the networks to converge. Only actions that have positive temporal difference (TD) errors are used to update the policy network, following the implementation of CACLA [32], while the Q network is updated regardless of the sign of their TD errors (line 15–22). Similar to [75], separate target networks are used to compute y_i to stabilize convergence. The weights of the target networks are updated to the latest

values in every 50 iterations.

Visuomotor Control

A visuomotor skill is the ability to synchronize visual information with physical movements. The visuomotor policy is a mapping from sensory images to actions. This is also called *End-to-End* control because it generates control outputs directly from raw observations. End-to-end visuomotor control is similar to how real animals sense the world and control their body.

We can use synthetic vision images taken from the viewpoint of the creature to simulate visuomotor skills. The creatures are assumed to be able to recognize obstacles from synthetic depth images (see Figure 5.13). The depth images are fed into Convolutional Neural Network (CNN) layers, which are connected to the front end of the policy/Q networks (see Figure 5.12). Although end-to-end learning of the deep neural networks is possible in principle using our algorithm, the actual computation of learning is substantial beyond the computing power of a single PC. The CNN has approximately 40k free parameters to learn, which is four times larger than the policy/Q networks.

A pragmatic solution is to learn visual perception separately from motor control and then end-to-end learning is used only to fine tune the whole visuomotor networks. Conceptually speaking, the creature first learns motor skills based on the geometric coordinates of the obstacles and then learns how the obstacles would look in the synthetic vision later. We used a small number of obstacles (ten in our experiment) to pre-train the policy/Q networks without CNN layers as explained so far. Each obstacle is represented by the coordinates of its center and the radius. The number of obstacles is fixed throughout the learning process because the number affects the structure of the neural networks. We then attach a virtual camera to the creature, and simulate the creature by using the pre-trained policy network in randomly initialized environments. Data (state, action, and value) are collected during the simulation, and then we use it for training the extended networks with CNN layers at the front end in

Figure 5.13 The synthetic vision images (64x64 pixels) from the viewpoint of the dragon in flight. The depth images were taken from the z-buffer of the OpenGL rendering pipeline.

a supervised learning fashion. In the learning process, only the weights of CNN layers are updated while the other parts of the networks remain intact. In our experiment, we collected 100,000 data and trained with 32 batch size.

Note that the motor policy using geometric coordinates does not generalize well if the new environment has more obstacles than the training environment. More importantly, the coordinate-based encoding is order-dependent, meaning that different numberings of obstacles would generate different sensory inputs regardless of its actual geometric configurations. This issue is not present with pixel-based vision inputs. Even though visuomotor learning takes the geometric coordinates as a supervisor, the visuomotor skill scales better with the generalization of the environment conditions. Figure 5.14 shows the performance comparison of control policies at the presence of progressively more obstacles. The visuomotor policy (green plot) deals better with dense obstacles than the motor policy with coordinate inputs (blue plot), because there is no prior assumption in visual perception as to the structure of obstacles.

5.3.4 Experimental Results

Our algorithm was implemented in Python, using DART [18] for the simulation of articulated body dynamics and TensorFlow [**Tensorflow**] for the training and evaluation of deep neural networks. The dynamics and learning parameters are summarized

Creature	Dragon	Worm	Spore
DoFs (actuable)	22	34	18
Mass (kg)	104	92	24
Length (meter)	5.0	5.2	7.4
Width (meter)	9.1	5.3	7.4
Height (meter)	0.6	0.4	8.6
dt (simulation)	0.001	0.001	0.001
dt (control)	0.2	0.2	0.2
Gravity (N/kg)	9.81	9.81	9.81
Density (ρ)	1.275	1.275	1.275
PD gains (wing)	10^5	10^5	10^4
PD gains (body)	5×10^4	2×10^4	N/A
Learning rate (π)	0.001	0.001	0.001
Learning rate (Q)	0.001	0.001	0.001
Discount factor (γ)	0.99	0.99	0.99
w_1 (target tracking)	0.01	0.01	0.01
w_2 (collision avoidance)	500	500	500
w_3 (effort minimization)	10^{-7}	10^{-8}	10^{-7}
w_4 (angular velocity)	0.5	0.5	0.5
w_5 (upright position)	20	20	20
w_6 (regularization)	0.05	0.03	0.05

Table 5.1 Creature details

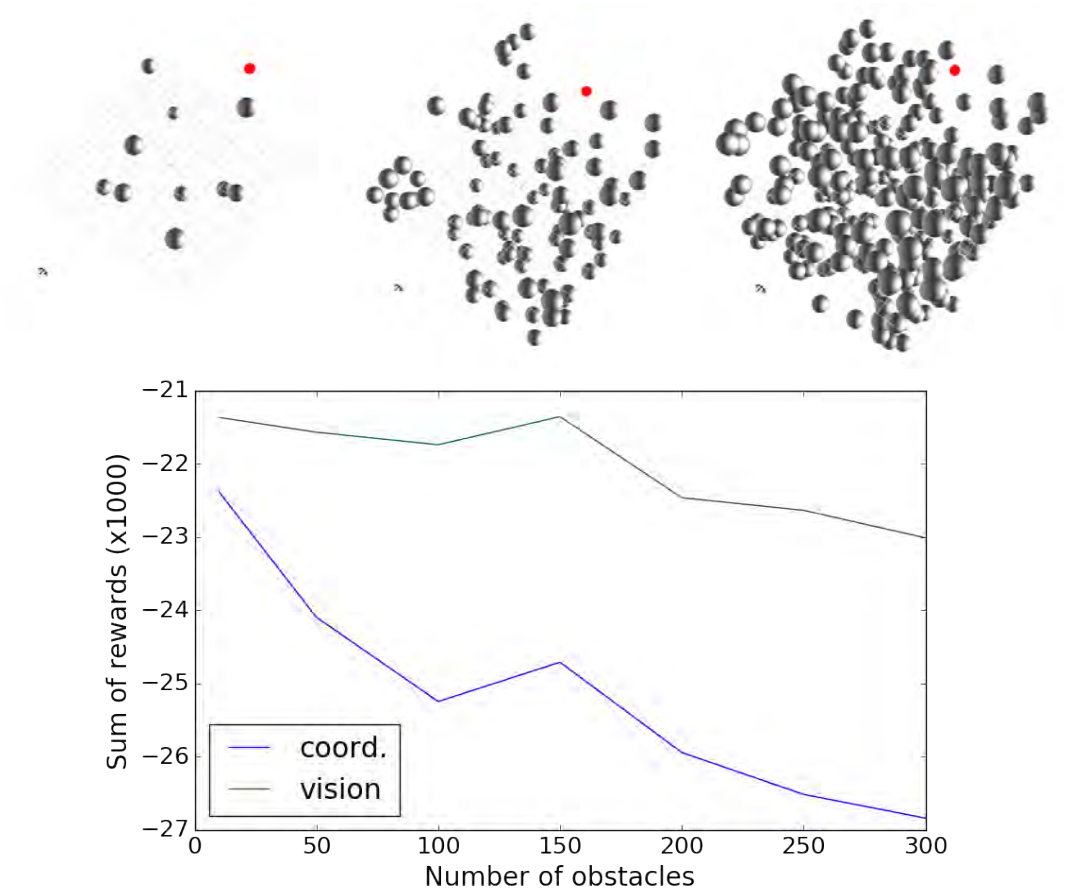


Figure 5.14 Visuomotor control experiments with the density of the obstacles. A red circle is the target and gray circles are the obstacles. We select the closest ten obstacles as inputs to the network using geometric coordinates, whereas such process is not needed for the visuomotor network.

in Table 5.1. The same values for simulation timestep (0.001 second), control timestep (0.2 second), gravity (9.81 N/kg), density (1.275), learning rate of π/Q (0.001), and discount factor (0.99) are used for all examples. The reward weight values for target tracking (w_1), collision avoidance (w_2), and upright position (w_5) are also fixed, whereas the weights for effort minimization (w_3), angular velocity (w_4), and regularization (w_6) depend on the choice of model and physics parameters. All processes

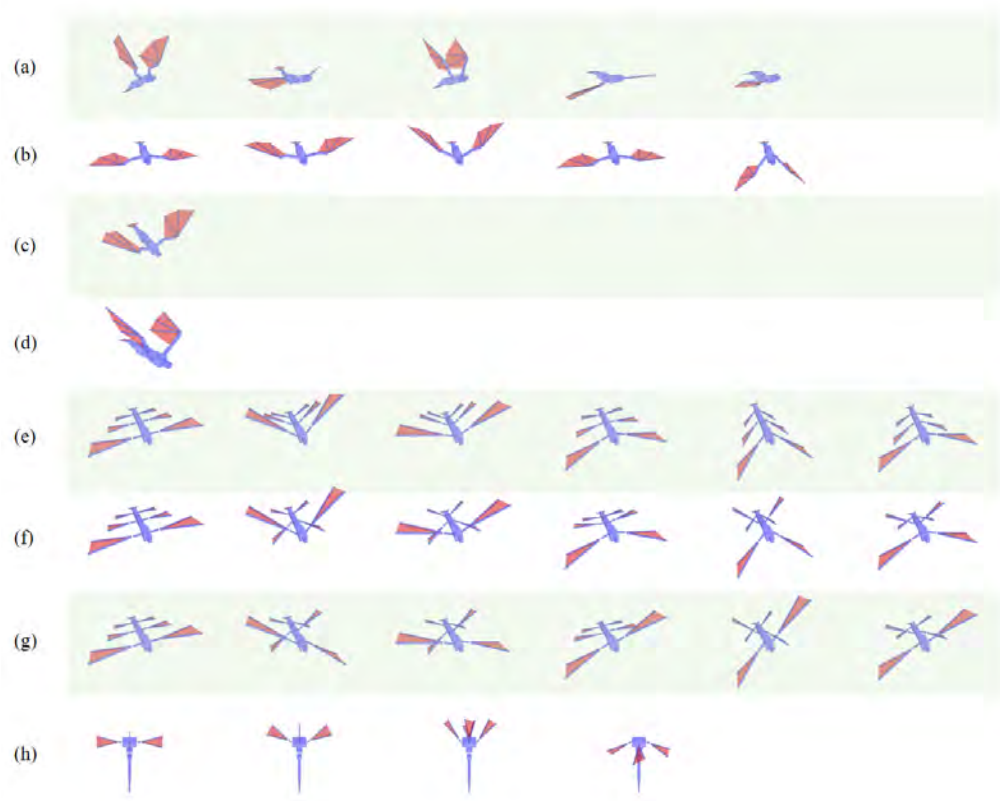


Figure 5.15 User-provided keyframes.

were run on a PC with Intel Core i7 6700K (4 cores, 4.0GHz). We did not utilize the computing power of GPUs because the computational cost is dominated by dynamics simulation rather than deep neural network operations.

Creature Models

We created three creature models. The dragon has a long trunk, two side wings, and a tail wing (see Figure 5.18). The trunk is composed of five body segments connected in a row. The side wings are attached at each side of the trunk, and the tail wing is attached at the end of the trunk. The side wings play a major role of propelling the body forward, while the tail wing supplements the function of side wings for fine maneuvers. The dragon is 5.0 meters long, 9.1 meters wide, and 0.6 meters tall.

We tested the dragon while varying its weights from 25 kg to 104 kg. The physical properties are inspired by an extinct species called *Pelagornis Sandersi*, which is known as the largest flying bird ever discovered. Our dragon of weight 104 kg is twice as big and three times heavier than the largest ever flying bird. The six-winged gigantic worm has three rows of wings and a body longer than the dragon. The four-winged spore is a light-weight flapping creature that can float easily in the air. The body density (mass per volume) of the spore is approximately three and six times lower than the dragon and the worm, respectively.

Dragon. Our dragon is provided with four sets of keyframes for energetic wingbeats of a large span, gentle wingbeats of a narrower span, gliding, and diving (see Figure 5.15 (a)-(d)). Interestingly, motor skills learned from different initial trajectories use different strategies to carry out a given task depending on the power per weight. The control policy of a light (25 kg) dragon using energetic wingbeats allows the dragon to maneuver rapidly and soar up easily towards the target, while the control policy of a heavy (104 kg) dragon using gentle wingbeats tends to make circular flying patterns and takes spiral paths to move upwards. Even though the gliding policy was learned using only one keyframe, the policy is still capable of steering its direction towards any target downwards. Similarly, the diving policy was also learned from a single keyframe, wherein the dragon folds its wings to dive fast. Switching between motor skills is immediate and effortless. The dragon can change its motor skill without any delay or preparation steps, because the neural network policies map any state to an action desired at the moment.

Gigantic Worm. The worm learned three control policies from the user-provided keyframes (see Figure 5.15 (e)-(g)). With the first policy, three rows of the wings flap synchronously in the same direction. With the second policy, the wings at even and odd rows flap in opposite directions. Alternatively, the worm twists its body to propel forwards rather than flapping its wings with the third policy. The worm learned all three motor skills successfully to stylize its locomotion patterns.

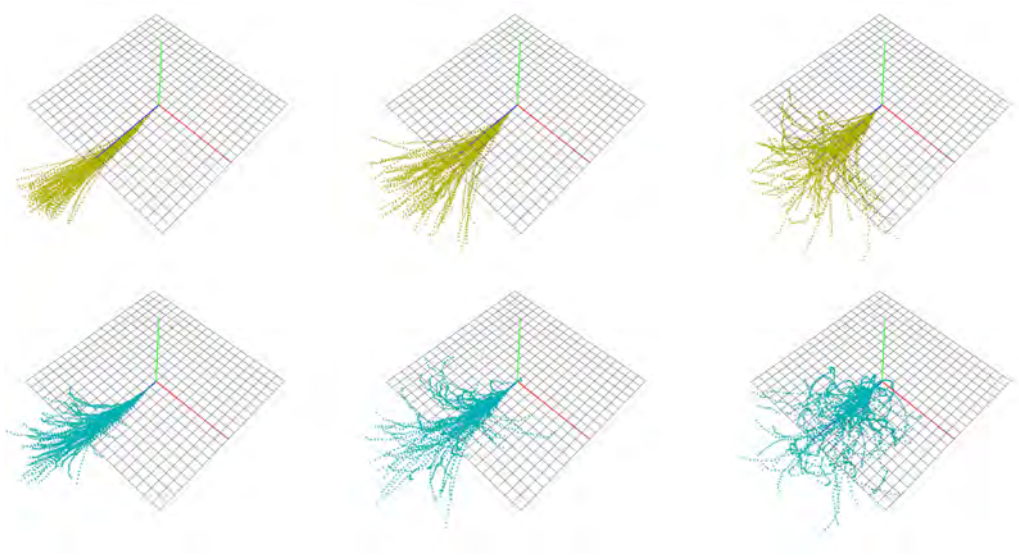


Figure 5.16 Robustness comparison. The magnitude of perturbation increases from left to right. (Top) The initial policy optimized with stochasticity. (Bottom) The initial policy optimized with standard CMA-ES.

Flapping Spore. Since all of its wings align radially facing upwards, moving up and down is easy for the spore, but it has to tilt the entire body to move horizontally. The locomotion style learned from the user-provided keyframes makes it spin around along the vertical axis for each wingbeat, resulting in a screw-like motion.

Learning

Constructing an initial control policy uses CMA-ES with its maximum generation of 100 and the population size of 8 at each generation. We sampled 10 random neighbors to estimate the expected rewards at each sample. Therefore, we run at most $100 \times 8 \times 10$ simulations, which is computationally demanding. Fortunately, individual episodic simulations are independent of each other and thus can be computed in parallel on CPU multi-cores. It takes about 10 to 20 minutes with four cores. Figure 5.16 shows the impact of stochastic optimization that minimizes expected rewards under uncertainty. Initial control policies were tested repeatedly at the presence ran-

dom perturbation. Scattering of the simulated trajectories indicates the robustness of the control policies. The optimized policy using the standard CMA-ES easily looses its balance and falls down even for small perturbation (Figure 5.16 (Bottom)). The optimization with stochasticity makes the wings stroke more energetically and thus resilient against perturbation (Figure 5.16 (Top)).

The evolutionary strategy can be paired with many existing policy update methods. Our algorithm in the previous section utilized the CACLA-style TD update because it worked well with flying creatures. Learning with evolutionary exploration takes about 25 hours to achieve the stable controller. Figure 5.17 shows the comparison of our evolutionary DQL with the base DQL. The plots indicate the convergence rates while the dragon learns its first control policy. The base DQL fell into a local minimum and failed to escape. Our evolutionary DQL not only converged faster but also discovered better control policy at the convergence. The difference is apparent in the supplementary video, wherein the dragon learned by our method is equipped with various motor skills that generate agile maneuvers. This agility cannot be achieved with the base DQL.

Benchmarks

To evaluate the effectiveness of our evolutionary strategy, we compared well-known DRL methods for continuous control with their evolutionary versions in OpenAI Gym [9]. The benchmark tests used three environments: *Swimmer-v1*, *HalfCheetah-v1*, and *HumanoidStandup-v1*. *Swimmer* is a two-dimensional snake-like creature with 8 observation dimensions representing its physical states and 2 action dimensions representing instantaneous joint torques. *HalfCheetah* is a two-dimensional two-legged creature with 17 observation dimensions and 6 action dimensions. The most challenging benchmark is *HumanoidStandup* with 376 observation dimensions and 17 actions dimensions. Its goal is to make a two-legged humanoid to stand up from the decubitus (lying) position.

Three base methods, CACLA [32], DDPG [99], and TRPO [95], were selected for

benchmarking. The performance of the RL methods for continuous control depends on the body structure, degrees of freedom, and actuation types of the creatures. TRPO is the current state-of-the-art in OpenAI Gym environments. We used the implementation of DDPG and TRPO by Duan et al [20]. We also implemented variants of CACLA and DDPG by plugging the evolutionary strategy, called Evo-CACLA and Evo-DDPG. Incorporating a new exploration strategy into TRPO is not straightforward because TRPO does not exploit replay buffers.

Figure 5.19 depicts the plots of average rewards for each environment. In our benchmark tests, both Evo-CACLA and Evo-DDPG consistently converged faster with better policies than their base methods for all environments. The advantage of exploiting the evolutionary strategy varies depending on the level of difficulty of learning. The simplest creature, the swimmer with two joints, benefits marginally from the smarter exploration strategy and thus the convergence depends largely on the choice of the policy update methods. The creatures with many joints benefit more to make substantial improvements. Evo-DDPG performs comparablely or better than TRPO in HalfCheetah and HumanoidStandup, though the convergence of Evo-DDPG is not as steady as the semi-monotonic convergence of TRPO. The convergence graphs of evolutionary methods fluctuate comparably to the graphs of their base methods. It means that our rapid evolution strategy does not cause instability beyond the characteristics of the base methods.

5.3.5 Discussion

Our physically simulated creatures learned how to fly using deep reinforcement learning. The key challenge was to have the policy and Q networks converge stably, while exploring unseen states and actions to generalize the capability of the initial motor skills. Achieving stable convergence and prompting rapid exploration are often contradictory goals. There are several sources of risks that may hinder stable convergence of the learning algorithm. First, theoretically, many reinforcement learning algorithms with general function approximators are not guaranteed to converge or

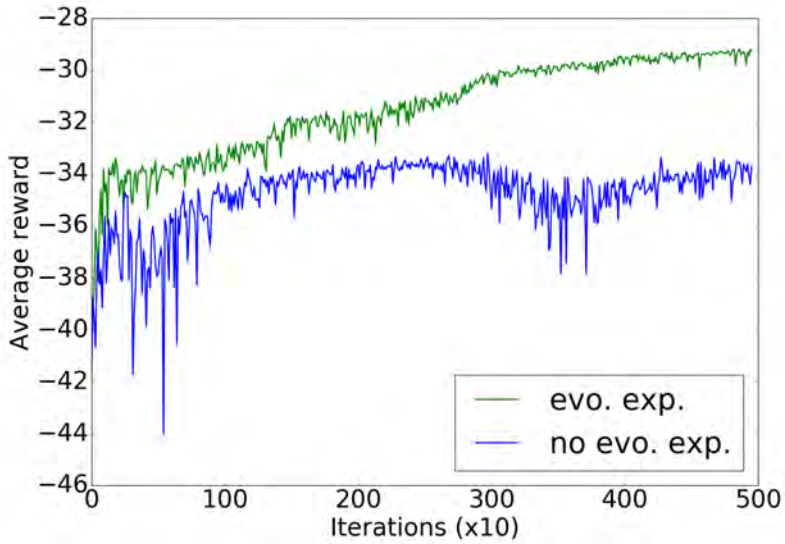


Figure 5.17 The convergence of DQL with and without evolutionary exploration.

Figure 5.18 Screenshots of flying creatures.

their convergence condition is not known yet. Therefore, careful parameter tuning is necessary in practice. Secondly, the forward dynamics simulation also has a notorious stability issue. The time integration would proceed stably only when the time step is sufficiently small. Using small time steps requires more computation for learning. Lastly, the balance recovery capability of the control policy is yet another source of instability that may impede exploration by perturbation. These seemingly-orthogonal issues are actually related with each other in practice. Assuming that the simulation time step is fixed, simulation and balance stability depend largely on the exploration rate, which is proportional to the magnitude of the random noise. Abrupt movements by jerky noise can make the simulation unstable and cause the creature to lose its

balance. Conversely, conservative settings for simulation would slow down the rate of exploration. Our evolutionary strategy addresses both issues, rapid exploration and simulation/balance stability, simultaneously.

Reinforcement learning would often discover new motor skills unexpected by the user. Our system also discovered unexpected motor skills, such as rapid turns by twisting the body and upside-down stunt flight, that are quite different from the user-provided keyframe animation. It is very difficult, if not impossible, to find such motor skills using trajectory optimization only because there are many local extrema between the initial trajectory and new motor skills. It is also difficult with DRL only because naïve random perturbation is too conservative to explore such far way goals. Leveraging both the exploration power of trajectory optimization and the generalization capability of DRL was a key to the successful discovery of new motor skills.

Some of previous studies on continuous control make use of parametrized action models, which represents a family of cyclic actions with a small number of parameters [42, 85]. The unit of action is a cycle of locomotion. Unlike those previous studies, we learn a mapping from state to action on a per-frame basis. Given a state, the control policy returns the desired actuation (joint angles) at the immediate moment. This nonparametric action model has advantages over the parametrized approaches. Most importantly, new motor skills may emerge, while parameterized approaches search policies within bounded action space. Since the parametrized action defines the phase of locomotion cycles, transitioning between control policies is usually allowed only at the beginning of a new cycle. In contrast, cyclic locomotion is an emergent behavior with nonparametric approach, and thus transitioning between control policies is allowed immediately regardless of the phase of locomotion.

The quality of the initial reference trajectory substantially influences the quality of the simulated motor skills. Currently, we used only five keyframes to design the initial trajectory, which may be not sufficient to describe natural looking motions. A remedy to the problem is the use of a realistic (either mocap or handcrafted by a professional artist) reference trajectory. Many RL methods for continuous control

generate stiff, jerky motions because the policy is learned based solely on rewards without any training data. Being able to learn control policies without training data is a double-edged sword. It seems leveraging minimal training data is necessary for better motion quality in graphics applications.

Even though our focus has been on flapping flight, our approach can be extended to deal with other types of locomotion, such as swimming, legged locomotion, and even limbless crawling. The strength of reinforcement learning lies mainly on its generalization capability that can deal with arbitrary body structures, sensors, actuators, motor skills, and environmental conditions.

There are many exciting directions for future research. Learning the sensorimotor skill of a skeletal model with musculotendon units requires a mapping from sensory inputs to muscle excitation signals [112, 59]. Deep learning would be a promising solution for muscle-based control considering the complexity of sensorimotor connection and the multiplicity of control layers. We are also interested in improving the simulation environment with more accurate aerodynamics simulation [105]. Turbulent air flow around the wings allows interesting maneuvers in bird flight, such as rapid break at large angle of attack. Such maneuvers cannot be learned without accurate simulation of turbulent wake and vortices. Learning other motor skills for flying creatures (take-off, landing, and flocking behavior) are also a challenging problem that has not been addressed by reinforcement learning yet.

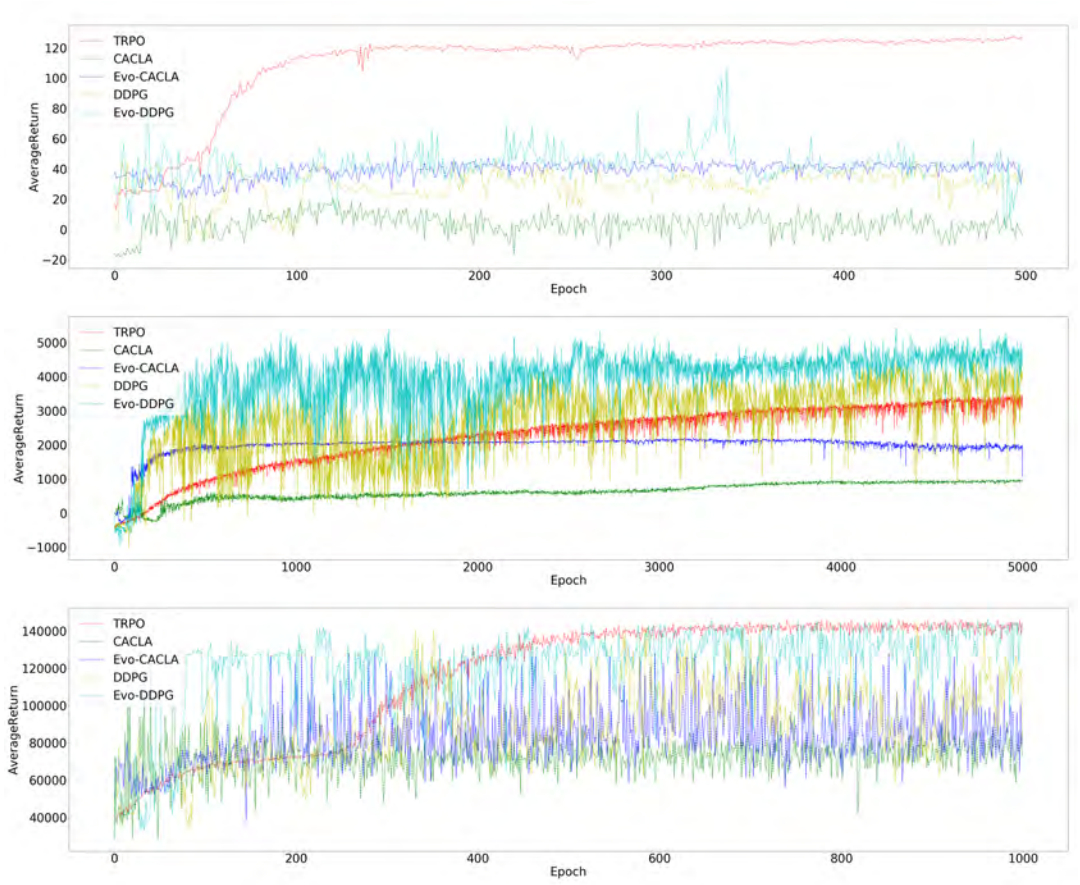


Figure 5.19 Benchmark results of Swimmer-v1 (Top), HalfCheetah-v1 (Middle), and HumanoidStandup-v1 (Bottom).

Chapter 6

Conclusion

6.1 Contribution

Choreographing motion is to convert written scripts, messages, or ideas to actual movements of actors. In performances or movie productions, directors spend a considerable time and effort because it is the most notable element by audiences. Choreographing multiple actor motion is a labor intensive work because space, time, causality, and mood coordinations should be satisfied to generate plausible motions. The thesis proposed computational approaches on choreographing multiple actor motion, where such coordinations can be satisfied without substantial effort.

We first introduced an approach for generating motions for a real performance. Shadow theatre, which requires unique stage settings and acrobatic motions, was demonstrated as an example performance and the results were comparable to ones made by the professional actors. The approach was to solve large optimization problems, where novel objective designs inspired by professional actors' strategies were suggested to narrow the huge search space and the derivative-free stochastic optimization was used. In the next, an interactive animation system for pre-visualization was introduced, which connects motion clips while satisfying the context of high-level

scene descriptions. Users can manipulate motions in the scene with in an interactive manner. Given scene descriptions for a scene, the system first selects appropriate combinations of motion clips, then synthesize them based on Laplacian motion editing technique. The cycle-based graph analysis for the description enabled that the resultant motions look plausible. The system then generates thousands of candidate scenes, where users can easily navigate the scenes and also re-rank them according to their preference. Scene ranking algorithm inspired by Google PageRank, which is based on our novel scene similarity, was proposed. Finally, we proposed two controller designs for physically simulated bird-like actors, who use flapping flight as primary locomotion. For the first method, efficient construction for a collection of trajectory optimization problems, which have goals assigned to grid cells, was introduced. For the second method, we suggested novel evolutionary deep Q-learning which exploits evolutionary strategy of CMA-ES for smart exploration.

6.2 Future Work

Densely Interacting Scene: There exist dense interactions in many performances, which means that individual actions are tightly tightly connected. For example, consider the script: 'Jack punches Tom. Tom dodges and kicks him back.' where the punch event is the direct cause of the dodge event and both events occur almost simultaneously. The dodge event causes another kick event immediately afterwards. Because this densely interacting scenario does not allow room for extended connections between subsequent events, proposed approaches in the thesis can not fully deal with the situation. One possible approach is to include the sequence in the motion capture session. Dealing effectively with long causal chains and dense interactions is a challenging goal for future research.

Computational Choreography for Other Components: Although the thesis focused on the motion of actors that is the most notable component in stage performances, there exist other components such as sound, lighting, props, and etc. For example, props such as tables or chairs could exist in the fight scene, then resultant

motions would entirely be changed. Because arrangement of the props decides the scenes, users should be able to consider it in the process of choreographing motion. To synthesize motions in such environments, we should first define that how to describe relationships and interactions between actors and props, and should be able to manipulate them as users want as shown in [44, 35]. We also could utilize physics simulation to generate diverse results [108, 30].

Open Animation System: Video production had been done by only professionals until personal video camera, or cell phones became popular. As the public can freely use video recording and editing techniques, they have started to make media themselves. Contents for entertainments have become more richer, and services such as YouTube, Twitch, and Vimeo have become influential than other traditional mass media as a result. Our computational approaches could be used to construct open animation system that the public could utilize to generate animations themselves. People could share their stories in an animation form similar to that we currently share photos or videos. Novice users could suggest an interesting performances or movies by showing a conceptual animation using an open animation system, for which the other public could invest through crowd funding platforms such as KickStarter.

Bibliography

[1] A. Agarwal and B. Triggs. "3D human pose from silhouettes by relevance vector regression". In: *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*. Vol. 2. 2004,

[2] Shailen Agrawal, Shuo Shen, and Michiel van de Panne. "Diverse Motion Variations for Physics-based Character Animation". In: *Proceedings of the 2013 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2013.

[3] Mazen Al Borno, Martin de Lasa, and Aaron Hertzmann. "Trajectory Optimization for Full-Body Movements with Complex Contacts". In: *IEEE Trans. Visualization and Computer Graphics* 19.8 (2013), pp. 1405–1414.

[4] Dragomir Anguelov et al. "SCAPE: Shape Completion and Animation of People". In: *ACM Transactions on Graphics*. SIGGRAPH '05 (2005), pp. 408–416.

[5] Arthur Appel. "Some Techniques for Shading Machine Renderings of Solids". In: *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*. AFIPS '68 (Spring). 1968, pp. 37–45.

[6] Okan Arikan, David A. Forsyth, and James F. O'Brien. "Motion Synthesis from Annotations". In: *ACM Transactions on Graphics (SIGGRAPH 2003)* 22.3 (2003), pp. 402–408.

[7] Amit Bermano et al. “ShadowPix: Multiple Images from Self Shadowing”. In: *Comp. Graph. Forum* (2012), pp. 593–602.

[8] Sergey Brin and Lawrence Page. “The Anatomy of a Large-Scale Hypertextual Web Search Engine”. In: *Computer Networks* 30.1-7 (1998), pp. 107–117.

[9] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

[10] T. W. Calvert and S. H. Mah. “Choreographers as Animators: Systems to support composition of dance”. In: *Interactive Computer Animation*. Ed. by N. Magnenat-Thalmann and D. Thalmann. Prentice-Hall, 1996, pp. 100–126.

[11] Justine Cassell, Hannes Högni Vilhjálmsson, and Timothy Bickmore. “BEAT: The Behavior Expression Animation Toolkit”. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. 2001, pp. 477–486.

[12] Byungkuk Choi et al. “SketchiMo: Sketch-based Motion Editing for Articulated Characters”. In: *ACM Trans. Graph.* 35.4 (2016), 146:1–146:12.

[13] Myung Geol Choi et al. “Deformable Motion: Squeezing into Cluttered Environments”. In: *Computer Graphics Forum (Eurographics 2011)* 30.2 (2011), pp. 445–453.

[14] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. “Robust Task-based Control Policies for Physics-based Characters”. In: *ACM Trans. Graph. (SIGGRPAH 2009)* 28.5 (2009).

[15] Stelian Coros et al. “Deformable Objects Alive!” In: *ACM Transactions on Graphics (SIGGRAPH 2012)* 31.4 (2012).

[16] Stelian Coros et al. “Locomotion skills for simulated quadrupeds”. In: *ACM Transactions on Graphics (SIGGRAPH 2011)* 30.4 (2011).

[17] Franklin C. Crow. “Shadow Algorithms for Computer Graphics”. In: *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’77. 1977, pp. 242–248.

[18] Dart. *Dart: Dynamic Animation and Robotics Toolkit*. `https://dartsim.github.io/`. 2012.

[19] James Davis et al. "A Sketching Interface for Articulated Figure Animation". In: *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '03. 2003, pp. 320–328.

[20] Yan Duan et al. "Benchmarking Deep Reinforcement Learning for Continuous Control". In: *CoRR* abs/1604.06778 (2016).

[21] CarlHenrik Ek, PhilipH.S. Torr, and NeilD. Lawrence. "Gaussian Process Latent Variable Models for Human Pose Estimation". In: *Machine Learning for Multimodal Interaction*. Vol. 4892. 2008, pp. 132–143.

[22] J. Funge, X. Tu, and D. Terzopoulos. "Cognitive Modeling: Knowledge, reasoning and planning for intelligent characters". In: *Proceedings of SIGGRAPH '1999*. 1999, pp. 29–38.

[23] Ran Gal et al. "3D Collage: Expressive Non-realistic Modeling". In: *Proceedings of the 5th International Symposium on Non-photorealistic Animation and Rendering*. NPAR '07. 2007, pp. 7–14.

[24] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10)*. 2010.

[25] Cindy M. Goral et al. "Modeling the Interaction of Light Between Diffuse Surfaces". In: *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '84. 1984, pp. 213–222.

[26] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey E. Hinton. "NeuroAnimator: Fast Neural Network Emulation and Control of Physics-based Models". In: *Proceedings of International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1998)*. 1998, pp. 9–20.

[27] P. Guan et al. “Estimating human shape and pose from a single image”. In: *Int. Conf. on Computer Vision, ICCV*. 2009, pp. 1381–1388.

[28] Martin Guay, Marie-Paule Cani, and Rémi Ronfard. “The Line of Action: An Intuitive Interface for Expressive Character Posing”. In: *ACM Transactions on Graphics* 32 (2013), 205:1–205:8.

[29] Sehoon Ha and C. Karen Liu. “Iterative Training of Dynamic Skills Inspired by Human Coaching Techniques”. In: *ACM Trans. Graph.* 34.1 (2014).

[30] Sehoon Ha et al. “Physics Storyboards”. In: *Computer Graphics Forum (Eurographics 2013)* 32 (2013), 133142.

[31] N. Hansen and A. Ostermeier. “Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation”. In: *Proceedings of IEEE International Conference on Evolutionary Computation*. 1996, pp. 312–317.

[32] Hado van Hasselt and Marco A. Wiering. “Reinforcement Learning in Continuous Action Spaces”. In: *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)*. 2007, pp. 272–279.

[33] Alejo Hausner. “Simulating Decorative Mosaics”. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’01. 2001, pp. 573–580.

[34] Nicolas Heess et al. “Learning Continuous Control Policies by Stochastic Value Gradients”. In: *Annual Conference on Neural Information Processing Systems (NIPS 2015)*. 2015, pp. 2944–2952.

[35] Edmond S. L. Ho, Taku Komura, and Chiew-Lan Tai. “Spatial relationship preserving character motion adaptation”. In: *ACM Transactions on Graphics (SIGGRAPH 2010)* 29.4 (2010).

[36] Ludovic Hoyet, Rachel McDonnell, and Carol O'Sullivan. "Push It Real: Perceiving Causality in Virtual Interactions". In: *ACM Transactions on Graphics (SIGGRAPH 2012)* 31.4 (2012).

[37] Kyunglyul Hyun et al. "Tiling Motion Patches". In: *IEEE Transations on Visualization and Computer Graphics* 19.11 (2013), pp. 1923–1934.

[38] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. "As-rigid-as-possible Shape Manipulation". In: SIGGRAPH '05. 2005, pp. 1134–1141.

[39] Eakta Jain, Yaser Sheikh, and Jessica Hodgins. "Leveraging the Talent of Hand Animators to Create Three-dimensional Animation". In: *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '09. 2009, pp. 93–102.

[40] Henrik Wann Jensen. "Global Illumination Using Photon Maps". In: *Proceedings of the Eurographics Workshop on Rendering Techniques '96*. 1996, pp. 21–30.

[41] Yushi Jing and Shumeet Baluja. "VisualRank: Applying PageRank to Large-Scale Image Search". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30 (2008), pp. 1877–1890.

[42] Eunjung Ju et al. "Data-driven control of flapping flight". In: *ACM Trans. Graph.* 32.5 (2013), 151:1–151:12.

[43] Jinwook Kim. *Virtual Physics: The realtime dynamics simulation library*. `http://virtualphysics.imrc.kist.re.kr/`. 2012.

[44] Manmyung Kim et al. "Synchronized multi-character motion editing". In: *ACM Transactions on Graphics (SIGGRAPH 2009)* 28.3 (2009), pp. 1–9.

[45] Manmyung Kim et al. "Tiling Motion Patches". In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '12. 2012, pp. 117–126.

[46] Lucas Kovar and Michael Gleicher. “Automated Extraction and Parameterization of Motions in Large Data Sets”. In: *ACM Trans. Graph.* 23.3 (2004), pp. 559–568.

[47] Lucas Kovar, Michael Gleicher, and F. Pighin. “Motion Graphs”. In: *ACM Transactions on Graphics (SIGGRAPH 2002)* 21.3 (2002), pp. 473–482.

[48] Paul G. Kry, Doug L. James, and Dinesh K. Pai. “EigenSkin: Real Time Large Deformation Character Skinning in Hardware”. In: *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’02. 2002, pp. 153–159.

[49] Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. “Feature-based locomotion controllers”. In: *ACM Transactions on Graphics (SIGGRAPH 2010)* 29.4 (2010).

[50] Jehee Lee. “A Hierarchical Approach to Motion Analysis and Synthesis for Articulated Figures”. PhD thesis. Department of Computer Science, Korea Advanced Institute of Science and Technology, 2000.

[51] Jehee Lee. “Representing Rotations and Orientations in Geometric Computing”. In: *IEEE Computer Graphics and Applications* 28.2 (2008), pp. 75–83.

[52] Jehee Lee and Kang Hoon Lee. “Precomputing Avatar Behavior from Human Motion Data”. In: *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2004, pp. 79–87.

[53] Jehee Lee and Sung Yong Shin. “A Hierarchical Approach to Interactive Motion Editing for Human-like Figures”. In: *Proceedings of SIGGRAPH ’99*. 1999, pp. 39–48.

[54] Jehee Lee et al. “Interactive Control of Avatars Animated with Human Motion Data”. In: *ACM Transactions on Graphics (SIGGRAPH 2002)* 21.3 (2002), pp. 491–500.

[55] Kang Hoon Lee, Myung Geol Choi, and Jehee Lee. "Motion patches: building blocks for virtual environments annotated with motion data". In: *ACM Transactions on Graphics (SIGGRAPH 2006)* 26.3 (2006).

[56] Sung-Hee Lee, Eftychios Sifakis, and Demetri Terzopoulos. "Comprehensive Biomechanical Modeling and Simulation of the Upper Body". In: *ACM Transactions on Graphics* 28.4 (2009).

[57] Yongjoon Lee et al. "Motion Fields for Interactive Character Locomotion". In: *ACM Trans. Graph. (SIGGRPAH Asia 2010)* 29.6 (2010).

[58] Yoonsang Lee, Sungeun Kim, and Jehee Lee. "Data-driven biped control". In: *ACM Trans. Graph. (SIGGRAPH 2010)* 29.4 (2010).

[59] Yoonsang Lee et al. "Locomotion Control for Many-muscle Humanoids". In: *ACM Trans. Graph. (SIGGRAPH Asia 2014)* 33.6 (2014).

[60] Sergey Levine and Vladlen Koltun. "Learning Complex Neural Network Policies with Trajectory Optimization". In: *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*. 2014.

[61] Sergey Levine et al. "End-to-end Training of Deep Visuomotor Policies". In: *J. Mach. Learn. Res.* 17 (2016), pp. 1334–1373.

[62] Sergey Levine et al. "Gesture Controllers". In: *ACM Transactions on Graphics (SIGGRAPH 2010)* 29.4 (2010).

[63] Sergey Levine et al. "Space-time Planning with Parameterized Locomotion Controllers". In: *ACM Transactions on Graphics* 30.3 (2011).

[64] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *CoRR* abs/1509.02971 (2015).

[65] Juncong Lin et al. "A Sketching Interface for Sitting Pose Design in the Virtual Environment". In: *IEEE Transactions on Visualization and Computer Graphics* 18 (2012).

[66] Libin Liu, Michiel Van De Panne, and Kangkang Yin. “Guided Learning of Control Graphs for Physics-Based Characters”. In: *ACM Trans. Graph.* 35.3 (2016).

[67] Libin Liu et al. “Sampling-based Contact-rich Motion Control”. In: *ACM Trans. Graph.* 29.4 (2010).

[68] Libin Liu et al. “Terrain runner: control, parameterization, composition, and planning for highly dynamic motions”. In: *ACM Transactions on Graphics (SIGGRAPH Asia 2012)* 31.6 (2012).

[69] Wan-Yen Lo and Matthias Zwicker. “Real-time planning for parameterized human motion”. In: *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2008, pp. 29–38.

[70] Matthew Loper et al. “SMPL: A Skinned Multi-Person Linear Model”. In: *ACM Transactions on Graphics (SIGGRAPH Asia 2015)* 34.6 (2015).

[71] J. Marks et al. “Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation”. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. 1997, pp. 389–400.

[72] Oliver Mattausch, Takeo Igarashi, and Michael Wimmer. “Freeform Shadow Boundary Editing”. In: (2013), pp. 175–184.

[73] Qiaozhu Mei, Jian Guo, and Dragomir Radev. “DivRank: The Interplay of Prestige and Diversity in Information Networks”. In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2010, pp. 1009–1018.

[74] N. J. Mitra and M. Pauly. “Shadow Art”. In: *ACM Transactions on Graphics* 28.5 (2009).

[75] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518 (2015), pp. 529–533.

[76] Igor Mordatch and Emanuel Todorov. "Combining the benefits of function approximation and trajectory optimization". In: *In Robotics: Science and Systems (RSS 2014)*. 2014.

[77] Igor Mordatch, Emanuel Todorov, and Zoran Popović. "Discovery of Complex Behaviors Through Contact-invariant Optimization". In: *ACM Trans. Graph.* 31.4 (2012).

[78] Igor Mordatch, Emanuel Todorov, and Zoran Popović. "Discovery of complex behaviors through contact-invariant optimization". In: *ACM Trans. Graph. (SIGGRAPH 2012)* 29.4 (2012).

[79] Marius Muja. *FLANN, Fast Library for Approximate Nearest Neighbors*. `http://mloss.org/software/view/143/`. 2011.

[80] Tomohiko Mukai and Shigeru Kuriyama. "Geostatistical Motion Interpolation". In: *ACM Trans. Graph.* 24.3 (2005), pp. 1062–1070.

[81] Meinard Müller and Tido Röder. "Motion Templates for Automatic Classification and Retrieval of Motion Capture Data". In: *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2006, pp. 137–146.

[82] Michael Neff et al. "Gesture Modeling and Animation Based on a Probabilistic Re-creation of Speaker Style". In: *ACM Transations on Graphics* 27.1 (2008).

[83] Claudio Pedica and Hannes Högni Vilhjálmsson. "SPONTANEOUS AVATAR BEHAVIOR FOR HUMAN TERRITORIALITY". In: *Appl. Artif. Intell. (IVA Special Issue)* 24.6 (2010), pp. 575–593.

[84] Fabio Pellacini, Parag Tole, and Donald P. Greenberg. "A User Interface for Interactive Cinematic Shadow Design". In: *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '02. 2002, pp. 563–566.

[85] Xue Bin Peng, Glen Berseth, and Michiel van de Panne. “Terrain-adaptive Locomotion Skills Using Deep Reinforcement Learning”. In: *ACM Trans. Graph. (SIGGRPAH 2016)* 35.4 (2016).

[86] Xue Bin Peng and Michiel van de Panne. “Learning Locomotion Skills Using DeepRL: Does the Choice of Action Space Matter?” In: *CoRR* abs/1611.01055 (2016).

[87] Xue Bin Peng et al. “DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning”. In: *ACM Trans. Grap. (SIGGRAPH 2017)* 36.4 (2017).

[88] Ken Perlin and Athomas Goldberg. “Improv: A System for Scripting Interactive Actors in Virtual Worlds”. In: *Proceedings of SIGGRAPH 1996*. 1996, pp. 205–216.

[89] R. Poppe. “Evaluating Example-based Pose Estimation: Experiments on the HumanEva Sets”. In: *CVPR 2nd Workshop on Evaluation of Articulated Human Motion and Pose Estimation (EHuM2),* 2007.

[90] R. Poppe and M. Poel. “Comparison of silhouette shape descriptors for example-based human pose recovery”. In: *Automatic Face and Gesture Recognition, 2006. FGR 2006. 7th International Conference on*. 2006, pp. 541–546.

[91] Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. “Reconstructing 3D Human Pose from 2D Image Landmarks”. In: *Proceedings of the 12th European Conference on Computer Vision - Volume Part IV*. 2012, pp. 573–586.

[92] Alla Safonova and Jessica K. Hodgins. “Construction and Optimal Search of Interpolated Motion Graphs”. In: *ACM Transactions on Graphics* 26.3 (2007).

[93] Tom Schaul et al. “Prioritized Experience Replay”. In: *CoRR* abs/1511.05952 (2015).

[94] John Schulman et al. “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. In: *CoRR* abs/1506.02438 (2015).

[95] John Schulman et al. “Trust Region Policy Optimization”. In: *CoRR* abs/1502.05477 (2015).

[96] Jamie Shotton et al. “Real-time Human Pose Recognition in Parts from Single Depth Images”. In: *Commun. ACM* 56 (2013), pp. 116–124.

[97] Hubert P. H. Shum, T. Komura, and S. Yamazaki. “Simulating Multiple Character Interactions with Collaborative and Adversarial Goals”. In: *IEEE Transactions on Visualization and Computer Graphics* 99 (2010).

[98] Hubert P. H. Shum et al. “Interaction patches for multi-character animation”. In: *ACM Transactions on Graphics (SIGGRAPH ASIA 2008)* 27.5 (2008).

[99] David Silver et al. “Deterministic Policy Gradient Algorithms”. In: *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*. 2014, pp. 387–395.

[100] C. Sminchisescu and A. Telea. “Human pose estimation from silhouettes, a consistent approach using distance level sets”. In: *In WSCG International Conference on Computer Graphics, Visualization and Computer Vision*. 2002.

[101] SNU-DB. *Seoul National University Motion database*. `http://mrl.snu.ac.kr/~mdb/`.

[102] Kwang Won Sok, Manmyung Kim, and Jehee Lee. “Simulating biped behaviors from human motion data”. In: *ACM Transactions on Graphics (SIGGRAPH 2007)* 26.3 (2007).

[103] Matthew Stone et al. “Speaking with Hands: Creating Animated Conversational Characters from Recordings of Human Performance”. In: *ACM Transactions on Graphics (SIGGRAPH 2004)* 23.3 (2004), pp. 506–513.

[104] Jie Tan, Greg Turk, and C. Karen Liu. “Soft Body Locomotion”. In: *ACM Transactions on Graphics (SIGGRAPH 2012)* 31.4 (2012).

[105] Jie Tan et al. “Articulated swimming creatures”. In: *ACM Transactions on Graphics (SIGGRAPH 2011)* 30.4 (2011).

[106] Jie Tan et al. “Learning Bicycle Stunts”. In: *ACM Trans. Graph. (SIGGRAPH 2014)* 33 (2014), 50:1–50:12.

[107] Adrien Treuille, Yongjoon Lee, and Zoran Popović. “Near-optimal Character Animation with Continuous Control”. In: *ACM Trans. Graph. (SIGGRAPH 2007)* 26.3 (2007).

[108] Christopher D. Twigg and Doug L. James. “Many-worlds Browsing for Control of Multibody Dynamics”. In: *ACM Transactions on Graphics (SIGGRAPH 2007)* 26.3 (2007).

[109] Kevin Wampler and Zoran Popović. “Optimal gait and form for animal locomotion”. In: *ACM Transactions on Graphics (SIGGRAPH 2009)* 28.3 (2009).

[110] Kevin Wampler et al. “Character animation in two-player adversarial games”. In: *ACM Transactions on Graphics* 29.3 (2010).

[111] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. “Optimizing Walking Controllers for Uncertain Inputs and Environments”. In: *ACM Transactions on Graphics (SIGGRAPH 2010)* 29.4 (2010).

[112] Jack M. Wang et al. “Optimizing Locomotion Controllers Using Biologically-Based Actuators and Objectives”. In: *ACM Transactions on Graphics (SIGGRAPH 2012)* 31.4 (2012).

[113] Xiaolin Wei and Jinxiang Chai. “Intuitive Interactive Human-Character Posing with Millions of Example Poses”. In: *IEEE Comput. Graph. Appl.* 31 (2011), pp. 78–88.

[114] Turner Whitted. “An Improved Illumination Model for Shaded Display”. In: *Commun. ACM* 23 (1980), pp. 343–349.

[115] Lance Williams. “Casting Curved Shadows on Curved Surfaces”. In: *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’78. 1978, pp. 270–274.

[116] B Y Philip C Withers. “An aerodynamic analysis of bird wings as fixed aerofoils”. In: *Journal of Experimental Biology* 90 (1981), pp. 143–162.

[117] Jia-chi Wu and Zoran Popović. “Realistic modeling of bird flight animations”. In: *ACM Transactions on Graphics (SIGGRAPH 2003)* 22.3 (2003), pp. 888–895.

[118] Yuting Ye and C. Karen Liu. “Synthesis of Detailed Hand Manipulations Using Contact Sampling”. In: *ACM Transactions on Graphics (SIGGRAPH 2012)* 31.4 (2012), 41:1–41:10.

[119] Barbara Yersin et al. “Crowd patches: populating large-scale virtual environments for real-time applications”. In: *Proceedings of symposium on Interactive 3D graphics and games*. 2009, pp. 207–214.

[120] Kangkang Yin, Kevin Loken, and Michiel van de Panne. “SIMBICON: Simple Biped Locomotion Control”. In: *ACM Transactions on Graphics (SIGGRAPH 2007)* 26.3 (2007).

[121] KangKang Yin et al. “Continuation methods for adapting simulated skills”. In: *ACM Transactions on Graphics (SIGGRAPH 2008)* 27.3 (2008).

[122] Q. Yu and D. Terzopoulos. “A Decision Network Framework for the Behavioral Animation of Virtual Humans”. In: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2007, 119–128.

요약

동작 연출은 스토리나 혹은 메시지를 전달하기 위한 배우의 움직임을 만드는 과정이다. 공연이나 영화에서, 관객들은 배우의 움직임 하나하나에 집중하기 때문에 감독은 이를 위해 상당한 시간과 노력을 들인다. 만약 한 장면에서 여러 명의 배우가 등장한다면, 동작 연출은 훨씬 더 어려워진다. 왜냐하면, 배우들 간의 시간, 공간, 인과관계, 분위기 등에 대한 합을 맞춰야하기 때문이다. 따라서 많은 감독들은 여러 배우들이 동작을 시각적으로 확인할 수 있는 스토리보드 혹은 간단한 애니메이션을 사용하는데, 감독들은 생각을 정리하거나 혹은 배우들에게 자신의 생각을 전달하는데 이들을 사용한다. 그러나 이러한 도구들을 사용하기 위해서는 예술적 감각은 물론 상당한 연습이 필요하다. 또한 수동적인 도구이기 때문에, 연출 과정 중에 실수나 혹은 더 나은 방법이 있더라도 어떠한 제안이나 피드백을 받을 수 없다. 마지막으로, 장면에 등장하는 배우들의 수가 늘어남에 따라서 필요한 수작업의 양이 기하급수적으로 증가하는 단점이 있다.

본 학위 논문은 컴퓨터를 활용하여 여러 사람의 동작 연출을 보조하는 방법들을 제안한다. 궁극적인 목적은 제안된 방법들을 이용하여, 초보자가 어렵지 않게 여러 배우의 동작을 생성할 수 있는 시스템을 만드는 것이다. 이를 위한 첫 번째 예제로, 다수의 배우가 하나의 목적을 위해 협동해야 하는 그림자 연극을 위한 동작 생성방법을 제안하고자 한다. 제안된 방법을 사용하면, 실제 전문 배우가 만든 동작들에 견줄만한 동작을 만들 수 있다. 두 번째 예제로는, 사용자가 장면에 대한 설명을 그래프 기반 인터페이스를 사용하여 만들면, 이에 대응되는 동작을 자동으로 생성하는 사전가시화용 상호작용 애니메이션 시스템을 제안한다. 마지막으로, 생성되는 동작의 물리적 타당성을 보장하기 위해, 배우를 물리적으로 모델링하여 시뮬레이션 하고 이를 위한 제어기를 설계하는 두 가지 접근 방법을 제안한다.

주요어: 그래픽스, 캐릭터 애니매이션, 여러 사람, 연출, 저작, 물리기반 제어, 딥러닝, 강화학습

학번: 2011-20881